

**ONTOLOGY CONSTRUCTION FOR THE HARMONIZED COMMODITY
DESCRIPTION AND CODING SYSTEM USING NATURAL LANGUAGE
PROSESSING TECHNIQUES**

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY
BY
FUNDA AKGÜR FAL**

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF MODELING AND DESIGN OF ENGINEERING
SYSTEMS
(MAIN FIELD OF STUDY: SOFTWARE ENGINEERING)**

JULY 2016

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

Prof.Dr. İbrahim Akman

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

Prof.Dr.Abdülkadir Erden

Head of Department

This is to certify that we have read the thesis “Ontology Construction for the Harmonized Commodity Description and Coding System Using Natural Language Processing Techniques” submitted by “Funda Akgür Fal” and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

Asst.Prof.Dr. Çiğdem Turhan

Supervisor

Co-Supervisor

Examining Committee Members

Asst.Prof.Dr. Çiğdem Turhan

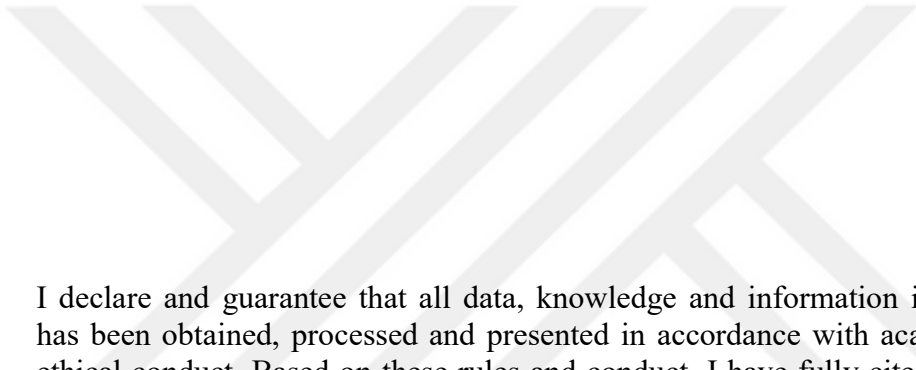
Asst.Prof.Dr. Reza Hassanpour

Asst.Prof.Dr. Altan Özkil

Asst.Prof.Dr. Erol Özçelik

Asst.Prof.Dr. Gökhan Şengül

Date:



I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Funda AKGÜR FAL

Signature

ABSTRACT

ONTOLOGY CONSTRUCTION FOR THE HARMONIZED COMMODITY DESCRIPTION AND CODING SYSTEM USING NATURAL LANGUAGE PROSESSING TECHNIQUES

Akgür Fal, Funda

Ph.D., Modeling and Design of Engineering Systems Department

Supervisor: Asst.Prof.Dr. Çiğdem Turhan

July 2016, 128 pages

World Custom Organization (WCO) is an interdependent intergovernmental organization whose mission is to improve the effectiveness and efficiency of customs administrations. WCO provides a shared code system for foreign trade namely, Harmonized Commodity Description and Coding System (HS) which is an international trade classification system including 5000 different code numbers. This system is used by custom's officers and traders who may or may not be expert users. The main problem of the existing system is that HS Code determination process can only be done by experts and not by non-specialists.

Ontologies are strong tools for representing the semantic structure. However, constructing large scale ontologies is hard and time consuming. Using Natural Language Processing (NLP) is an efficient way to construct ontologies.

In this study, the main goal is to develop an ontology for HS codes using Turkish natural language processing. The constructed ontology is expected to provide three vital benefits for users in determining the correct HS code. Since the constructed ontology defines the HS code with respect to the definition of the products, any user will be able to identify any products' HS code using Turkish language. Second, using an ontology behind the system provides flexibility and cost efficiency where all the

products and their relations are processed and recognized by the ontologies' natural framework. Third, in the literature there have not been any studies that have developed such a search system for Turkish customs.

This thesis provides new perspective for ontology construction and searching using Turkish NLP that is a promising novel method and a flexible approach for any querying systems.

Keywords: Ontology, Natural Language Processing, Protégé, Harmonized Commodity Description System.



ÖZ

ARMONİZE MAL TANIMI VE KODLAMA SİSTEMİ İÇİN DOĞAL DİL İŞLEME KULLANARAK ONTOLOJİ OLUŞTURULMASI

Akgür Fal, Funda

Doktora, Mühendislik Sistemlerinin Modellenmesi ve Tasarımı Bölümü

Tez Yöneticisi: Yar.Doç.Dr.Çiğdem Turhan

Temmuz 2016, 128 sayfa

Dünya Gümrük Örgütü (DGÖ) gümrük idarelerinin etkinliğini ve verimliliğini artırmayı hedef edinmiş hükümetler arası bir kuruluştur. DGÖ dış ticaret için ortak bir kod sistemi sağlamaktadır. Armonize Mal Tanımı ve Kodlama Sistemi (Armonize Sistem-AS) diye adlandırılan bu sistem, 5000 farklı kod numarası içeren uluslararası bir ticari sınıflandırma sistemidir. Sistem uzman ya da uzman olmayan gümrük memurları ve ticaret yapmak isteyen insanlar tarafından kullanılmaktadır. Mevcut sistemin temel sorunu; AS kodu tespiti işlemlerinin ancak konu uzmanları tarafından yapılabilmesi ve uzman olmayan kişilerce yapılamamasıdır.

Ontolojiler semantik yapıyı gösteren güçlü araçlardır. Büyük ölçekli ontolojilerin yapılandırılması zor ve zaman alıcıdır. Doğal Dil İşleme (DDİ) kullanımı ontolojileri oluşturmak için etkili bir yoldur.

Bu çalışmada, ana amaç AS kodları için Türkçe doğal dil işleme yapılarak ontoloji oluşturmaktır. Oluşturulan ontolojinin AS kodlarının tespit edilmesinde üç hayati fayda sağlaması beklenmektedir. Oluşturulan ontolojide AS kodları ürünlerin tariflerinden yola çıkarak tanımlanmış, böylece uzman olmayan kişilerin Türkçe kullanarak AS kodlarını tespit etmeleri sağlanmıştır. İkinci olarak, sistemi bir

ontoloji üzerine kurmak ontolojilerin doğal yapıları sayesinde, ürünler ve aralarındaki ilişkilerin tanımlanmasıyla sistemin esneklik ve düşük maliyet ile çalışmasını sağlamaktadır. Son olarak, literatürde, Türk gümrüklerinde kullanılan böyle bir arama sistemi bulunmamaktadır.

Bu tez, ontoloji oluşturma ve Türkçe dil işleme kullanarak aramada yeni bir bakış açısı sağlarken, oldukça kullanışlı yeni bir metod ve esnek bir arama sistemi yaklaşımı sunmaktadır.

Anahtar Kelimeler: Ontoloji, Doğal Dil İşleme, Protégé, Armonize Mal Tanımı ve Kodlama Sistem



To My Dear Family,

ACKNOWLEDGEMENTS

I would like to express sincere appreciation to my supervisor Asst. Prof.Dr. ıgdem Turhan for her guidance, supportive and constructing approach throughout the research. My graditude to you is beyond the words.

I would like to thank to my thesis jury members Prof. Dr. Alok Mishra and Asst. Prof. Dr. Reza Hassanpour for their guidance in evaluation and review of my study.

I would also like to thank to my managers at work for understanding me during my study.

Grateful thanks to my parents, Yunus and Semiha Akgür and my brother Cem Akgür for their endless support, love and understanding throughout my life. I appreciate your presence.

I am also grateful to my dear husband Mehmetcan Fal, for his continuous support, encouragement, patience and technical insight during this period. He is the best motivation for me to not give up.

TABLE OF CONTENTS

ABSTRACT	iii
ÖZ	v
ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xv
CHAPTER 1	1
INTRODUCTION	1
1.1 Overview	1
1.2 Contributions of the Thesis	3
1.3 Organization	4
CHAPTER 2	5
LITERATURE REVIEW.....	5
2.1 Background on Ontologies.....	5
2.1.1 Semantic Web	5
2.1.2 Ontologies	6
2.1.3 Ontology Tools	18
2.1.4 Reasoners	25
2.1.5 Knowledge Access	31
2.2 Background on Natural Language Processing	37
2.2.1 Natural Language Processing (NLP)	37
2.2.2 Turkish NLP	45
2.2.3 Ontology Construction Using NLP Techniques	48
CHAPTER 3	50
ANALYSIS AND DESIGN OF THE SYSTEM.....	50
3.1 Ontology Design.....	50

3.1.1	Analysis of Domain Data	51
3.1.2	Modeling and Construction of Ontology	54
3.2	Studies on NLP	62
3.2.1	Keyword Extraction using NLP	62
3.2.2	Keyword Injection.....	66
CHAPTER 4	67
IMPLEMENTATION OF THE SYSTEM	67
4.1	Implementation.....	67
4.1.1	Keyword Creation Using Turkish NLP.....	69
4.1.2	G.T.İ.P. Search Application	82
CHAPTER 5	91
EVALUATION OF THE SYSTEM	91
5.1.	Evaluation of Ontology	91
5.2.	Query Response Time Comparison	99
CHAPTER 6	102
CONCLUSION AND DISCUSSION	102
Future Work	105
REFERENCES	106
APPENDICES	115
APPENDIX A	115
APPENDIX B	116
APPENDIX C	117
APPENDIX D	119
APPENDIX E	122
APPENDIX F	125

LIST OF TABLES

Table 1. Some examples from OWL vocabulary.....	17
Table 2. Logical representation of <i>apple</i>	40
Table 3. Chapter list of the Section 11	52
Table 4. Product Groups list of Chapter 50 (İpek).....	52
Table 5. Product list of product group 5007	53
Table 6. Example of morphologically analyzed keywords.....	63
Table 7. Morphologically analyzed word examples.	64
Table 8. Morphologically analyzed word examples.	65
Table 9. Query response time for different number of keywords in seconds.	99
Table 10. Query result for 1 keyword	101
Table 11. Query result for 3 keywords.....	101
Table 12. Query result for 5 keywords.....	101
Table 13. Query result for 3 keywords with negative component search.....	101
Table 14. Query result for 100 queries.....	101

LIST OF FIGURES

Figure 1. Illustration of single ontology architecture.....	8
Figure 2. Illustration of multiple ontology architecture.....	9
Figure 3. Illustration of hybrid ontology approach.....	10
Figure 4. Illustration of ontology levels.....	11
Figure 5. Sowa Diamond a top level ontology illustration.	12
Figure 6. Illustration of Top-Down Vs Bottom-Up Approach.....	13
Figure 7. Illustration of the Web Stack.....	17
Figure 8. Illustration of architecture of Collaborative Protégé.....	21
Figure 9. Classes in Ontology in Protégé Ontology Editor.....	22
Figure 10. Example of Relations in the Protégé Ontology Editor.....	23
Figure 11. Comparison of the ontology development tools.....	24
Figure 12. Illustration of Pellet architecture.....	27
Figure 13. Illustration of Hermit architecture.....	29
Figure 14. Comparison of Reasoners.....	30
Figure 15. Reasoners comparison graphs.....	31
Figure 16. Jena components.....	32
Figure 17. Illustration of Jena Layers.....	33
Figure 18. OWL API design illustration.....	34
Figure 19. Illustration of DL Query.....	36
Figure 20. Morphological analysis of words in Turkish.....	40
Figure 21. Syntactic structure of Sentence1.....	41
Figure 22. Syntactic structure of Sentence2.....	42
Figure 23. A morphological analysis example using.....	47
Figure 24. Position of Chapter 50 in HS coding system.....	51
Figure 25. Class hierarchy graph of <i>İpek</i> class.....	56

Figure 26. Class hierarchy graph of <i>İpek</i> class-2	567
Figure 27. Class hierarchy of Ontology from Protégé.	58
Figure 28. Individual/Class type definition.....	59
Figure 29. Ontology structure graph-1	60
Figure 30. Ontology structure graph-2.....	61
Figure 31. Ontology structure graph-3 / <i>İpek</i> Chapter hierarchy	62
Figure 32. Components of the system.....	68
Figure 33. Data flow diagram of Keyword Creation application.....	69
Figure 34. Affixes and their special definitions in the application.	71
Figure 35. An example of morphological analysis of the words	72
Figure 36. Data property definitions of individuals.....	74
Figure 37. Object property definitions of individuals.....	75
Figure 38. An individual definition and its data properties	76
Figure 39. Keywords defined as data property	77
Figure 40. Anahtar class and its members.	78
Figure 41. An example of Equivalent class definition.....	80
Figure 42. As an example of SubClassof restriction.....	81
Figure 43. Data Flow Diagram of Application	83
Figure 44. Data Flow Fragment of <i>Parser</i>	83
Figure 45. Data Flow Fragment of <i>QueryBuilder</i>	85
Figure 46. Pseudo Code of <i>QueryBuilder</i>	86
Figure 47. Data Flow Fragment of <i>QueryEngine</i>	87
Figure 48. Data Flow Fragment of <i>Printer</i>	87
Figure 49. Sequence diagram of <i>Printer</i>	89
Figure 50. Pseudo code of <i>Printer</i>	90
Figure 51. Query result of target product queried with keyword <i>ipek</i>	93
Figure 52. Query result of target product queried with keywords <i>ipek</i> and <i>döküntü</i> . 94	
Figure 53. Query result of target product for keywords <i>ipek</i> , <i>döküntü</i> and <i>iplik</i>	95
Figure 54. Query result of target product for keywords <i>ipek</i> , <i>döküntü</i> , <i>iplik</i> and <i>koza</i> 96	
Figure 55. Query result of target product queried with keywords <i>ipek</i> , <i>döküntü</i> , <i>iplik</i> , <i>koza</i> and <i>misina</i>	97
Figure 56. Query result of the product using <i>-siz</i> suffix.....	98

Figure 57. Query result of the product using *olmayan* keyword..... 99

Figure 58. Illustration of keywords query for different number of keywords 100



LIST OF ABBREVIATIONS

HS	-	Harmonized Commodity Description and Coding System
WCO	-	World Customs Organization
NLP	-	Natural Language Processing
G.T.İ.P	-	Gümrük Tarife İstatistik Pozisyonu
OWL	-	Web Ontology Language
API	-	Application Programming Interface
GUI	-	Graphical User Interface
NL	-	Natural Language
URI	-	Uniform Resource Identifier
W3C	-	World Wide Web Consortium
DGÖ	-	Dünya Gümrük Örgütü
AS	-	Armonize Sistem
RDF	-	Resource Description Framework
RDF-S	-	Resource Description Framework Schema
DL	-	Description Logic
AI	-	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 Overview

World Custom Organization (WCO) is an interdependent intergovernmental organization whose mission is to improve the effectiveness and efficiency of customs administrations (World Customs Organization, 2015). The Harmonized Commodity Description and Coding System (HS) is the most important tool for standardization of the international trade. HS is used by WCO member countries as international trade classification system that provides common codes for all tradable goods. Member country's HS code is the same as the code of other member countries.

Individuals and institutions engaged in foreign trade need to have HS code of the target product, therefore determining correct code of the good of interest is extremely important. In order to own the HS code, the subject product has to be known by the customs officers. Moreover, it is also important for the traders to import or export the product.

Giving or finding an exact code for a desired product is not an easy task. In order to define a code, customs officers have to be familiar with the product and product characteristics. This kind of determination or classification is difficult to generate for non-expert officers or non-expert traders.

The HS system currently consists of 5000 groups of products consisting of twelve digits. The first six-digit codes belong to the product is identical for all countries, whereas the last six digits define the product for each country differently.

The existing system does not meet the needs of non-specialists to determine the correct HS Code for a given product. Currently, HS Code determination process can only be done by experts. Therefore, it is necessary to develop a system to improve the efficiency and accuracy of HS Code determination process. Additionally, the system should eliminate the need of human intervention.

In order to satisfy the goals above, such a system should accomplish the following minor goals. This system has to keep all the products' code and all the products description; must relate each description to its own unique code. Furthermore, this system needs to store the relationships between the products properly. Note that, in the HS Code system products are being represented in a hierarchical structure. Therefore, utilizing an ontology for representing HS Code system can prove to be efficient and appropriate.

The idea of the ontologies emerged with the Semantic Web. The focus of the Semantic Web is, developing a technology for linking data in meaningful manner. In short, an ontology is a hierarchically constructed set of concepts (Blomqvist et al., 2006). As a state of the art, there are number of searching mechanisms based on ontologies which are seen as the next generation of search engine tools. Therefore, using an ontology is the better way to develop a content classification application. Using an ontology in the searching mechanism supports an improvement for keyword-searching with the help of class hierarchy, rules and relationships.

Since HS Code system represents a number of related products, the products are obviously linked based on their simple linguistic descriptions. Based on this framework, ontology construction using natural language processing techniques is proposed for the thesis. There are number of theories and techniques that researchers have studied for developing a system which automates ontology construction with the help of NLP techniques. Since HS Code system is a set of entities described by a

natural language, constructing an ontology using NLP techniques will provide a better way to reach the mentioned minor goals.

1.2 Contributions of the Thesis

In this study, a system is developed to terminate the human intervention for querying process of HS codes. Application of this thesis will contribute to research on Turkish NLP as well, since in previous research, there has not been any example for constructing an ontology using NLP for the Turkish language. This thesis hopefully is expected to bring a new perspective to Turkish researchers focusing on searching mechanisms.

In fact, the developed system has a few numbers of instances for the maintenance of customs' information in several countries. However, all of these systems have different coding strategies. This divergence with respect to coding occurs because of the language which is used for the description of domain data. In detail, the reasons of divergence are because of the followings:

- Each language has its own semantic structure,
- Languages can have different syntactic structures,
- There are number of morphological architectures.

Therefore, these types of systems diverge generally at the point of language analysis of the target domain. Parsing, finding the related roots and understanding the semantic relation of the words can only be done by language analysis. Consequently, similar examples to this study can actually be completely different than each other. From this aspect, this study is entirely novel and proposes a framework for Turkish querying systems.

This study has a number of contributions to literature. Firstly, this is an original and a new approach for ontology construction with NLP for the Turkish language. Moreover, the developed application is an appropriate and effective tool to improve the search quality of the users.

Also, this thesis is a promising study to reveal new development trends especially for similar needs.

1.3 Organization

The organization of the thesis in the following chapters is as follows:

Chapter 2 provides background information about ontologies. Moreover, different ontology tools, different perspectives and basics of ontology are presented. Additionally, this chapter gives related information on natural language processing where all the information about Turkish NLP tools is explained.

Chapter 3 presents information about understanding HS code domain and designing NLP techniques on this domain.

In Chapter 4, the implementation of the application is given. The system architecture and the queries supported by the system are described.

Chapter 5 includes the evaluation of the overall system, followed by Chapter 6 which concludes the thesis and provides some possible future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Background on Ontologies

Ontology is expressed as a formal representation of knowledge. This formal representation consists of a set of concepts within a domain and the relationship between these concepts (Subhashini & Akilandeswari, 2011).

2.1.1 Semantic Web

“I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A “Semantic Web”, which makes this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The “intelligent agents” people have touted for ages will finally materialize”

Tim Berners-Lee (1999)

“The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries” (“W3C Semantic Web Activity”, 2013). The goal of semantic web is, developing a technology for

linking data in an effective and meaningful manner. This technology deals with the knowledge, not the data itself. Semantic Web presents a network for defining, storing and representing the meaning of the data. Semantic Web also reasoning by using some rules for knowledge representation.

Knowledge representation is an essential need for knowledge-based systems. Yet, representation is also a challenging task for knowledge-based systems and the users. Semantic Web serves and organizes data in a machine understandable format. This format provides a way for sharing and using information between users and automated tools (Baader et al., 2001). To aid knowledge representation, semantic description languages/standards have been developed. Some of these languages are RDF, SPARQL, OWL, and SKOS which are used for the integration of data (“W3C Semantic Web Standards”, 2014).

Main focus of developing semantic web languages is, finding a point between the expression ability of a language and flexibility of reasoning services. Importantly, the point must be in balance (Gentner et al., 2012). In this aspect, ontologies play a central role for the development of semantic application. In order to provide a formal model for descriptive and semantic information technologies, ontologies are essential (Knublauch et al., 2004).

2.1.2 Ontologies

“An ontology is a hierarchically structured set of concepts describing a specific domain of knowledge that can be used to create a knowledge base. An ontology contains concepts, a subsumption hierarchy, arbitrary relations between concepts, and axioms. It may also contain other constraints and functions.”

(Blomqvist et al., 2006)

Ontologies define a shared vocabulary to share information in a specific domain (Noy & McGuinness, 2001). In the book by F. Baader (2003), ontologies are mentioned as description of logic. In addition, ontology is also expressed as a formal representation of knowledge. This formal representation consists of a set of concepts

within a domain and the relationship between these concepts (Subhashini & Akilandeswari, 2011). By constructing a model, providing a shared and common understanding is the main outcome of ontologies (Nguyen, 2011). The only limitation about the ontologies is the computational power. Domain wideness is only correlated by the computation power of the host system.

Basic Ontology Primitives

Following expressions are the logical categories of an ontology:

- Entities: every class, individual and property is accepted as an entity in an ontology,
- Classes: are set of collection, concepts, objects, etc.,
- Relationship: the relation between two entities,
- Attributes: properties, characteristics of an object,
- Restrictions: Described assertions,
- Axioms: rules that are described in logical form,
- Data range/domain: definition of an entity that can be represented in a class,
- Functions: stand for representing complex relations that can be used instead of a term in a statement,
- Rules: also statements that can identify logical inferences,
- Events: modification of relation or attributes.

These categories are used for logical description and these elements are interpreted and used for meaningful interpretation (Motik et al., 2009).

2.1.2.1. Ontology Architecture

In the process of ontology construction, an ontology should not over commit on representational choices. Ontologies should be extensible based on needs during the actual use. Organizing principle should be used to structure an ontology (Swartout et al., 1996). Wache and his friends' article (Wache et al., 2001) claims that there are

three general ontology construction architectures. These architectures can be identified as:

- Single ontology,
- Multiple ontology,
- Hybrid ontology.

Single Ontology

Single ontology approaches take into account only one global ontology. This type of ontology provides a shared lexicon for the determination of semantics. All the information origin is described by only one ontology. Furthermore, information providers can also be some other ontologies but there are important restrictions. In such an ontology, information provider must have nearly the same view of the domain. Consequently, the multiple ontology structure is developed to solve this integration problem (Wache, et al., 2001). Single ontology approach is depicted in Figure 1.

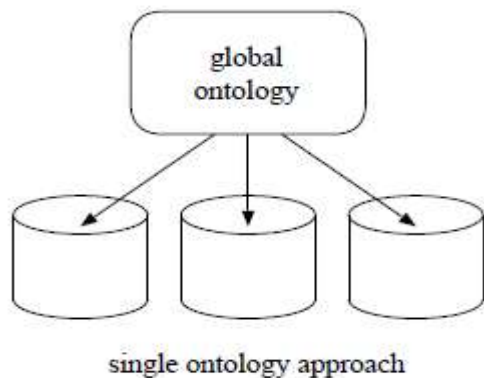


Figure 1. Illustration of single ontology architecture (Wache et al., 2001).

Multiple Ontology

In this type of ontology, each information origin is defined by its own ontology. This combination namely *source ontology* consists of several separate ontologies. In fact, for this perspective, information sources, other ontologies in this case are not

assumed to be ontologies. Instead of assuming these information sources as ontologies, whole structure can be seen as an ontology that shares the same vocabulary. Multiple ontology sources should have different views and details of a domain. In this perspective, each ontology in the system can be developed separately from each other. Developing new data sources or modifying the existing data sources are advantages. However, comparison of ontology sources with each other may cause problems in the multiple ontology approach (Wache et al., 2001).

Multiple ontology approach can be seen in Figure 2.

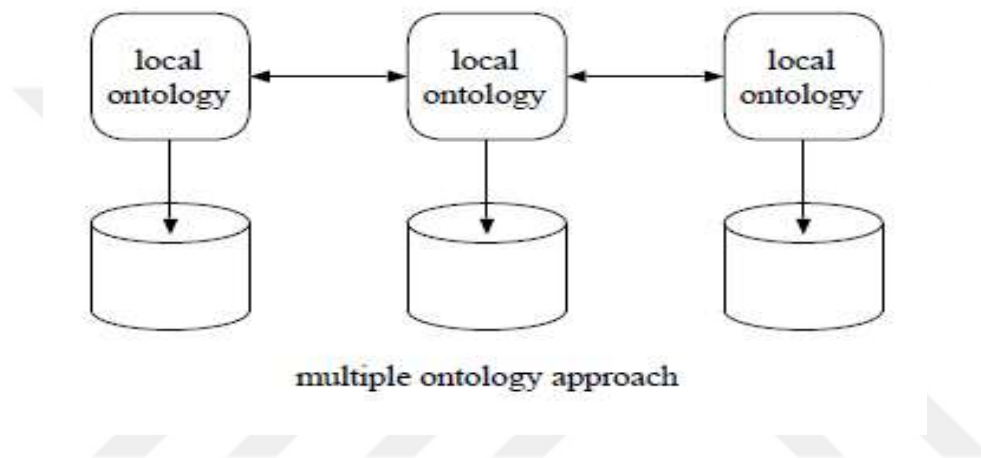


Figure 2. Illustration of multiple ontology architecture (Wache et al., 2001).

Hybrid Ontology

This approach was developed to get rid of the disadvantages of single and multiple ontologies. Although the structure is similar to the multiple ontology, the difference comes from the single ontology approach. In this architecture, vocabulary is sourced by only one global ontology (Goh, 1996). Hybrid approach is presented in Figure 3.

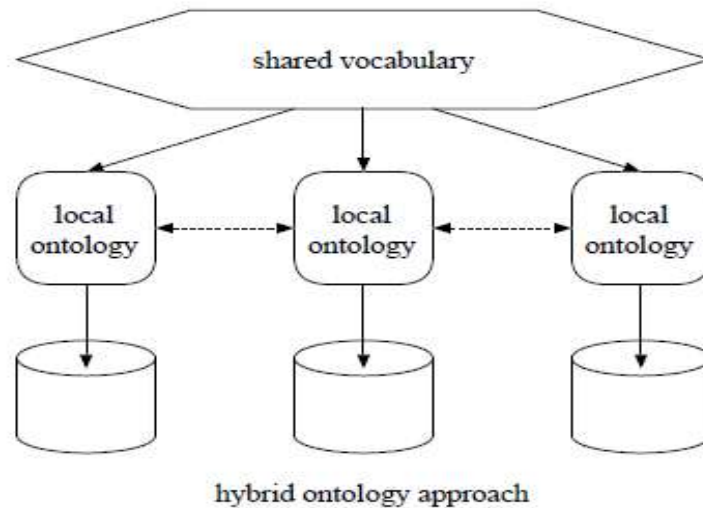


Figure 3. Illustration of hybrid ontology approach (Wache et al., 2001).

2.1.2.2 Ways of Constructing an Ontology

In the literature, there are mainly three ways to construct an ontology (Subhashini & Akilandeswari, 2011).

Manual ontology construction requires domain expertise, time and much effort. It is reliable, but difficult to construct.

Semi-automatic that is an ongoing process construction which means that the construction is done with human intervention (Jaimes & Smith, 2003). Since the developed application includes a semi-automatic ontology, detailed explanation of this construction approach and the reason of why this approach adapted to this study can be found in Chapter 4.

Fully-automatic construction is responsible for all the construction processes. In this type of construction, processes of constructing relations, injecting examples, keywords, etc are only performed by the ontology itself. Since semi-automatic construction approach also has an ability to construct an ontology, the difference with respect to fully-automatic ontology construction is that automatization is somehow done by identifying patterns and using these patterns to construct the ontology (Blomqvist, 2005).

2.1.2.3 *Ontology Conceptualization*

Ontology types can be summarized with respect to their conceptualizations. The following Figure visualizes the dimensions of levels.

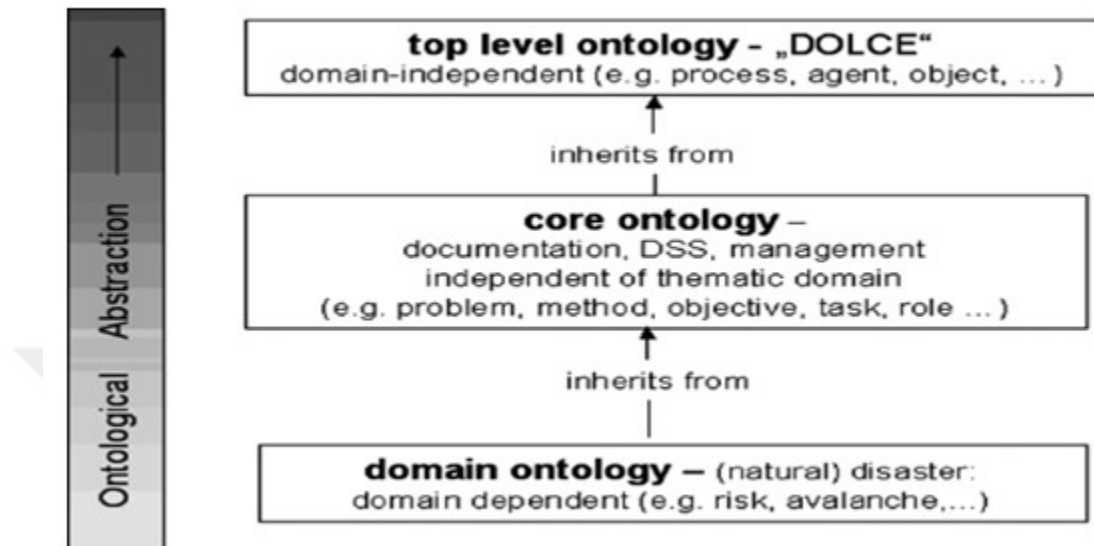


Figure 4. Illustration of ontology levels (Mayer et al., 2008).

Top level ontology: Concepts of ontology are independent from the domain. It is the highest level ontology. Top level ontologies can be used for getting answers to the following questions:

How can the top level be presented if this top level is a general class of all classifications?

What is the most general class?

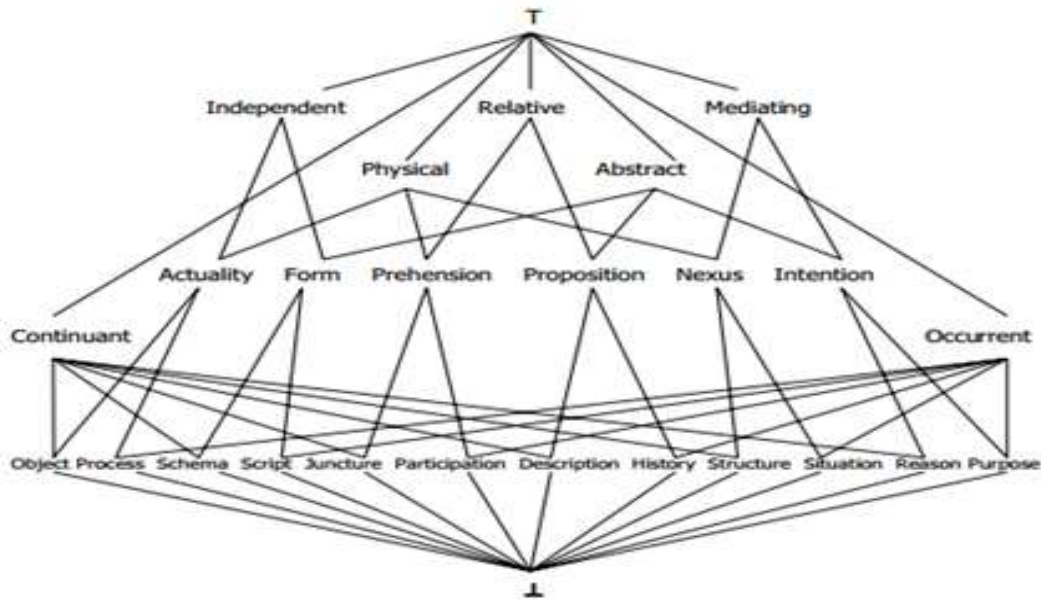


Figure 5. Sowa Diamond a top level ontology illustration (“Top-Level Categories”, 2001).

Generally, top level ontologies are constructed for representing general knowledge for common sense reasoning (Jansen, 2008). An example can be seen in Figure 5.

Mid level ontology: In this level, ontologies stand like a bridge between top level and domain ontologies. Mid level ontologies generally provides more concrete representations to the upper level ontologies. This concrete representation is abstraction of concepts found in upper ontology (Möller & Sintek, 2007).

Domain ontology: Concepts of ontology and their relationships describes the domain. It is the most common ontology. Basically, it describes concepts of domain of interest and their relations (Möller & Sintek, 2007).

Task ontology: Concepts of the ontology describes the structure of performing the tasks domain. It is designed for a specific task. These type of ontologies supports terms specifically for a particular task. For example, the word *hypothesis* belongs to the diagnosis task ontology (Fensel, 2001).

Application ontology: Concepts of this type of ontology are domain specific (Nguyen, 2011).

2.1.2.4 Ontology Construction Approaches

There are two more approaches in constructing ontologies. An ontology can be constructed according to the direction of knowledge extraction. The type of directions can be seen in Figure 6. According to the direction of knowledge extraction, these two construction approaches are called top-down and bottom-up. Figure 6 can perfectly depict these approaches.

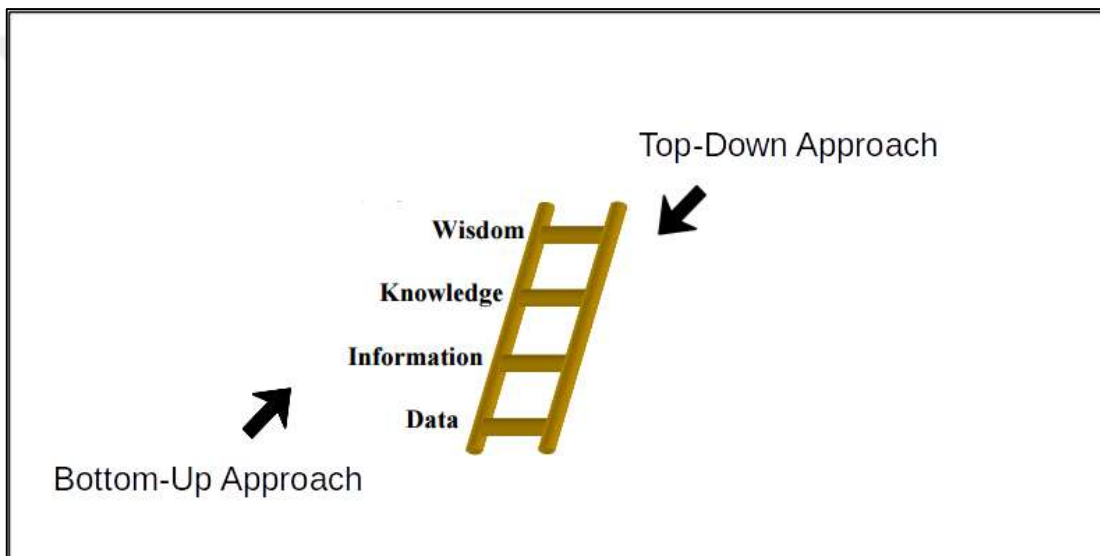


Figure 6. Illustration of Top-Down Vs Bottom-Up Approach (Gandon, 2011).

Top down approach can start with the definition of the top concept. Specialization also needs to be done at the starting point. In this approach, the highest level of an ontology includes the most generic concept (Gandon, 2011).

Bottom up approach is sourced by atomism that notion perceive every entity as composed of other entities. The lowest level of each object includes a set of indivisible atomic units and their relations. An ontology constructed by the bottom-up approach can be defined as a concept described by its examples. In this ontology, each example differs from its superconcept and also these examples differed by its sub concepts. Therefore it is obvious that descriptive atomism means that the bottom-

up approach also classifies any object by its own instance objects (Van & Mars, 1998).

Moreover, an ontology can be constructed according to the starting point of information extraction that is namely concept-driven or a data-driven approach.

Data-driven approach requires a specific domain data to construct ontologies and uses domain knowledge with human intervention. (Jaimes & Smith, 2003). As a data driven approach ontology, Public Procurement Services of Korea built an ontology based e-catalog system named KOCIS (Lee. et. al., 2006).

Concept-driven approach is for constructing ontologies with general domain knowledge, not with specific data. This type of ontologies allows the user to define specific internal structure, for example, type of data or particular conditions etc. (Dautov et al., 2012). In the concept-driven approach, the prior knowledge of the designer is vital (Swaak & Jong, 2001).

There exist a number of ontologies in the literature constructed by different approaches. In Bruckschen and his friends' study (Bruckschen et al., 2010); an ontology is constructed for domain specific knowledge in the legal domain. Moreover, Witte and his friends (Witte et. al., 2010) have constructed an ontology using NLP which can be considered as a good example for domain specific and fully automatic ontology. In this study, they have created a tool, OwlExporter, which was used to map entities, and also for analysing OWL ontologies.

In broad perspective, there is not an exact model in ontology construction, so there can be many different ontology designs even for the same data in any domain. This gives a freedom in ontology construction but it is also a confusing step for the ontology designers. Therefore, the constructor must decide about an efficient and reliable ontology structure according to the goal of the system but still considering the domain data.

Analysing present ontologies from different domains also provides better understanding for constructing a class structure and their relations. In this study,

many constructed ontologies from different domains have been analyzed to decide on a proper ontology construction, and developing an intuition for the current study. Some of the analyzed ontologies were Wine ontology (Noy & McGuinness, 2001), Pizza Ontology (“Pizza Ontology”, 2015), eClassOWL Product and Services Ontology (“eClassOWL”, 2015), Travel Ontology (Choi et al., 2006) and a similar Customs Ontology that China has developed, named CIQ2000, to facilitate inspection, quarantine and custom processing. (Zang et al., 2008).

2.1.2.5 *Ontology Construction Steps*

Ontologies are constructed in six basic steps as mentioned below:

- *identifying* the need and purpose source of the ontology,
- *capturing* the concepts and their relationships that need to be identified,
- *deciding* about the representational language for the ontology construction,
- *implementation* of the ontology,
- *ontology evaluation* that means each ontology must be evaluated with respect to an evaluation criterion,
- *ontology documentation* where all the related knowledge of the ontology has to be documented for reuse (Subhashini & Akilandeswari, 2011).

Furthermore, there are four main components of the ontologies that are used for representing a domain:

- *Concepts* can be classified as entities in a domain,
- *Relation* is the relationship between these concepts,
- *Instance* means instance of the concepts,
- *Axioms* that are representational axioms which are always true.

2.1.2.6 *Ontology Representation Languages*

Ontology languages can be classified as:

Frame-based languages (OKBC, Ontolingua, OCML, XOL, etc.) which represent objects and concepts as frames and build relationships of frames (Lenzerini et al., 2004),

Description Logics-based languages (KIF, OWL, DAM+OIL etc.) which represent concepts and classes as well as their relationships in a well-defined structure and in a basic way (Lenzerini et al., 2004),

XML-related languages (RDF, RDFS, etc.) use XML documents to represent data by using tags in XML syntax (Lenzerini et al., 2004).

The Resource Description Framework (RDF) is a structure that provides a way to encode, exchange and use the metadata again and again. RDF is an XML application that pushes data in a strict structural form. In addition, it provides certain methods for expressing semantics. This constrained and standardized form supports the interchange of metadata between different resource description communities (Miller, 1998).

Resource Description Framework Schema is used for declaring lexicon, semantic properties' type identified by a certain communities. Definition of RDF schemas stands for the valid properties of the provided RDF description (Miller, 1998).

The Web Ontology Language, OWL, is one of the most popular semantic mark-up languages that simply work on the World Wide Web for sharing and publishing information. OWL is developed by the W3C Web Ontology Working Group. The first level of RDF schema was not enough to extract meaningful information for the machines. Therefore, there had to be a language which can go beyond the RDF Schema. OWL simply was developed to satisfy this need. The basic growing structure which is named as Web Stack that carries OWL on its shoulder and Web Stack can be described in five bullets (also see Figure 7):

- XLM is the first level of Web Stack that is the surfaced syntax where the documents need to be structured to be processed. Note that at this level, there is nothing related to semantic constraints on these documents.

- XML Schema is a language that restricts the structure of XML documents.
- RDF is basically a data model for resources and relations between these resources.
- RDF Schema is used for identification of properties and classes of RDF objects.
- OWL is the extended vocabulary for describing classes and properties. In this vocabulary, there are number of specific expressions for defining relations and properties. Some of the examples are given in Table 1.

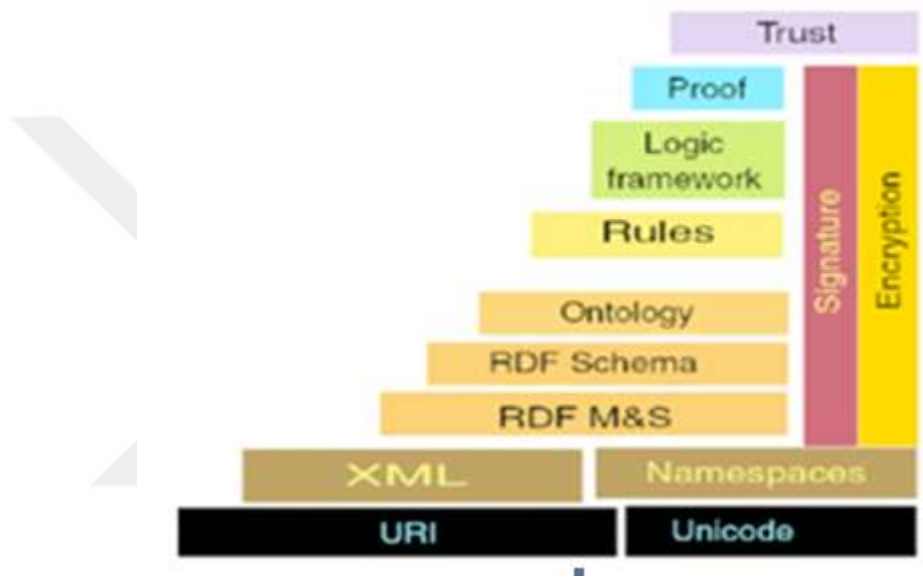


Figure 7. Illustration of the Web Stack (Horrocks et al., 2005).

Table 1. Some examples from OWL vocabulary.

Definition	Expression
Relations	Disjointness
	Union
	intersection of
Cardinality	Only
	Some
	exactly one

Table 1 (Continued).

Characteristic Property	Symmetry
	Transitive
	Functional

OWL is relatively powerful in expressibility; however, it is highly complex for computability. Under OWL there are three sub languages. These languages are:

- OWL Lite supports the most basic needs for classification and simple constraints. For example, there is only one definition for cardinality that can be defined by 0 or 1. OWL Lite is the simpler sub language of OWL family.
- OWL DL provides maximum expressivity. However, there are some restrictions about this expressivity. For instance, a class can be a subclass of many classes but it cannot be an instance of another class. Therefore, because of these description logics, this sub language of OWL is named as OWL DL.
- OWL Full can be used for maximum expressiveness and maximum syntactic flexibility of RDF. Nevertheless, there is no computational guarantee.

In this study, OWL DL is used for the querying of the constructed ontology. Although OWL DL is the most accurate and the most expressive sub language, it brings a number of restrictions for the usage of OWL. In OWL DL, all classes, properties, data types and instances need to be accepted as a pairwise disjoint (Zhang & Miller, 2005).

2.1.3 Ontology Tools

Since constructing an ontology for huge amount of data is time consuming and difficult, there are many powerful tools developed for ontology construction. The main goals of these tools are, providing an easy way for the construction of ontologies. There exist several types of ontology tools. The main concept is providing a system for building a new ontology from scratch or reuse of previously developed ontologies. The purpose can also be a merging and alignment of an ontology. In addition, it can be used as ontology evaluation tool, ontology-based

annotation tool, ontology querying service or an ontology learning tool.

With the strong capabilities of ontology construction tools, in ontology operations, developers and also domain experts have gained a powerful skill in ontology construction, manipulation and querying (Horridge et al., 2006).

These tools can be classified into two main groups as *language specific* and *language independent* ontology tools. Examples of these tools are given below:

Apollo

Apollo is the user friendly ontology tool. The modelling of the tool is based on fundamental primitives such as classes, relations, functions etc (Kapoor & Sharma, 2010). This application stands on the OKBC frame system that is the extension of the Frame Ontology. In this system knowledge organization used is frame-based representation (Gómez-Pérez et al., 2004)

Ontolingua

This system provides a portable ontology where multiple languages can be used for representation. Since the application converts the definitions into a basic syntax, these syntactic forms are taken as an input to the several implemented representation. Therefore, Ontolingua can be used by multiple users and groups so that they can choose their representational system and they can move it from one system to another system easily (Gruber, 1992).

WebOnto

WebOnto is an application that provides visualization, browsing and editing options for constructing and editing ontologies. This system models the information based on OCML which is a language that can be used for knowledge modelling, providing a representational system and knowledge bases. OCML is also developed using WebOnto itself (Domingue et al., 1999).

WebODE

This tool supports a way for the integration of several ontologies. Besides, it also provides ontology design, management, browsing and other standard ontology tool abilities. Within the system, middleware service of the tool stands for the integration of ontologies into information systems (Arpírez et al., 2001).

Protégé

As a language independent ontology tool, Protégé, is a popular tool. It is an open source ontology editor which is suitable for building intelligent systems. Protégé is developed at Stanford Centre of Biomedical Informatics and Research at the Stanford University School of Medicine. It is made available under the BSD 2-clause license. Protégé is defined in its web source as a tool for modeling ontologies in order to connect data integration with business supporting algorithms (“Protégé”, 2016). Protégé has a strong ability and knowledge modeling. The following features are the capabilities of Protégé:

- Loading, editing and saving OWL and RDF ontologies,
- Constructing ontologies,
- Storing ontologies in different formats such as XML, RDF and OWL,
- Graphical representation of ontologies,
- Populating ontologies,
- Reasoning ontologies. (“Protégé”, 2016)

Protégé is developed based on two different components, namely model components and view components. Model components were developed for ontology and knowledge representation that is processed in the background. It provides a Java based API. By the help of this API, querying and editing the ontology and its entities for OWL 2.0 can be performed. View component of Protégé is used for a user friendly graphical user interface. This interface supports the creation or manipulation of ontology entities. Via this interface, classes, individuals, objects, data properties and relations between them can be created and manipulated. Protégé also supports several formats such as CLIPS, RDF, XML, UML and relational database access.

Protégé users are also able to develop new features and plugins. Additionally, they can use these plugins integrated with the tool Protégé, dynamically (Knublauch et al., 2004). Protégé has many plugins and libraries for different purposes of ontology construction (“Protégé Plugins Library”, 2015). For instance, using the OWL plugin, Protégé gains the capability of editing ontologies by multiple users simultaneously. With reasoner plugins, Protégé has the capability of reasoning ontologies from the user interface. Moreover, for ontology visualization, several graph representation plugins can be used within Protégé (Knublauch et al., 2004). In order to export ontologies, there is an ontology export plugin.

There are several ways of constructing and storing an ontology using Protégé. Designers can either build their ontologies using Frames, RDF(S) and OWL and can also store the constructed ontologies into databases or files. With Protégé's capability in interoperability, Protégé can also be used in a client/server architecture or stand alone architecture. The client–server architecture of Collaborative Protégé is depicted in Figure 8:

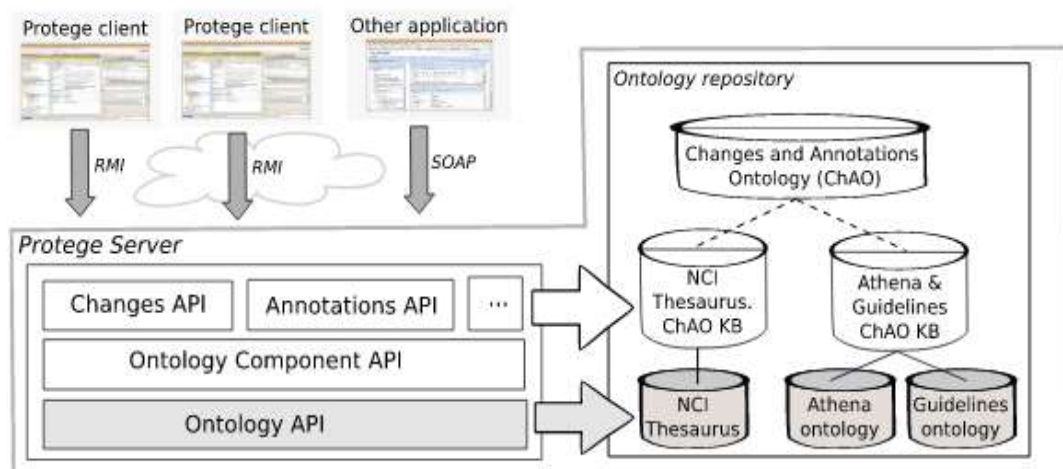


Figure 8. Illustration of architecture of Collaborative Protégé (Tudorache et al., 2008).

Protégé environment is in a tabular style that supports users to define entities, classes, individuals, object and data properties, annotation properties, relations and the types of relations. The environment also supports ontology reasoning.

Furthermore, it supports many other abilities with its downloadable plugins (Saripalle et al., 2013). The Protégé ontology construction environment is illustrated in Figures 9 and Figure 10.

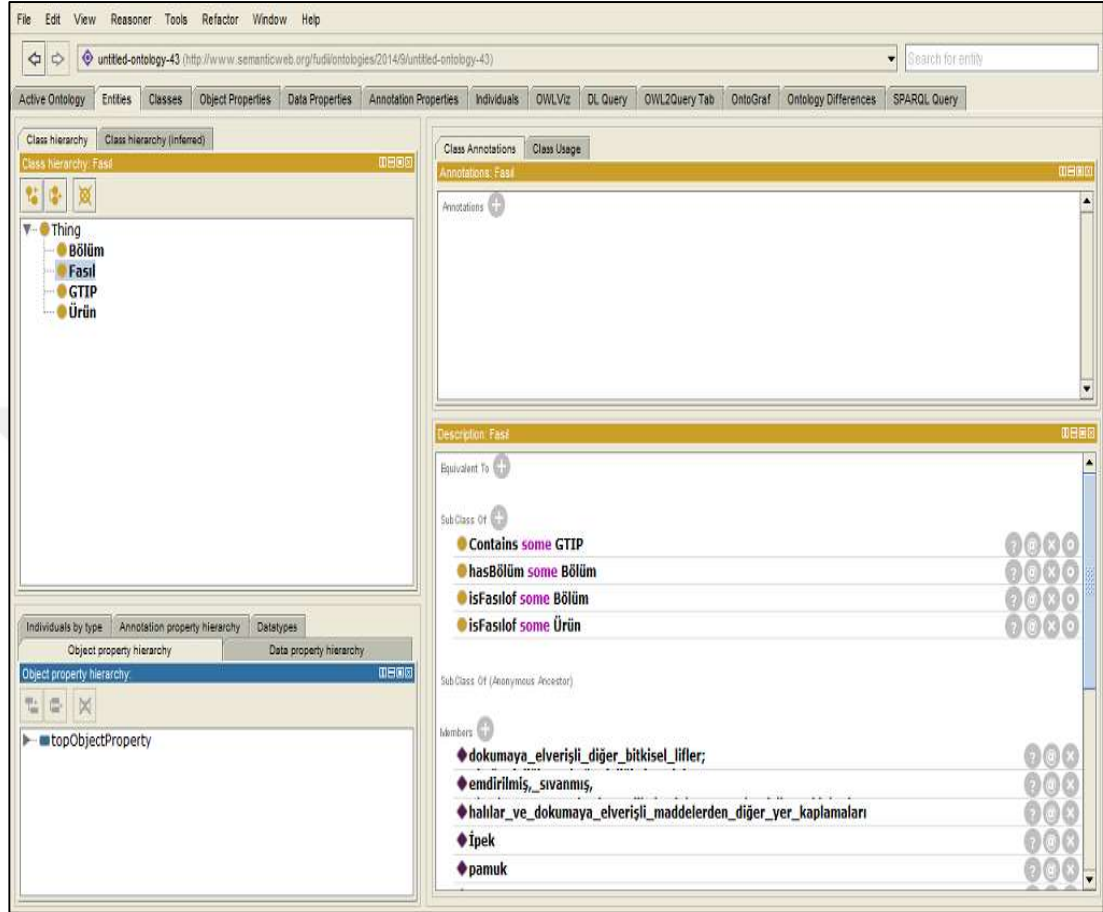


Figure 9. Classes in Ontology in Protégé Ontology Editor.

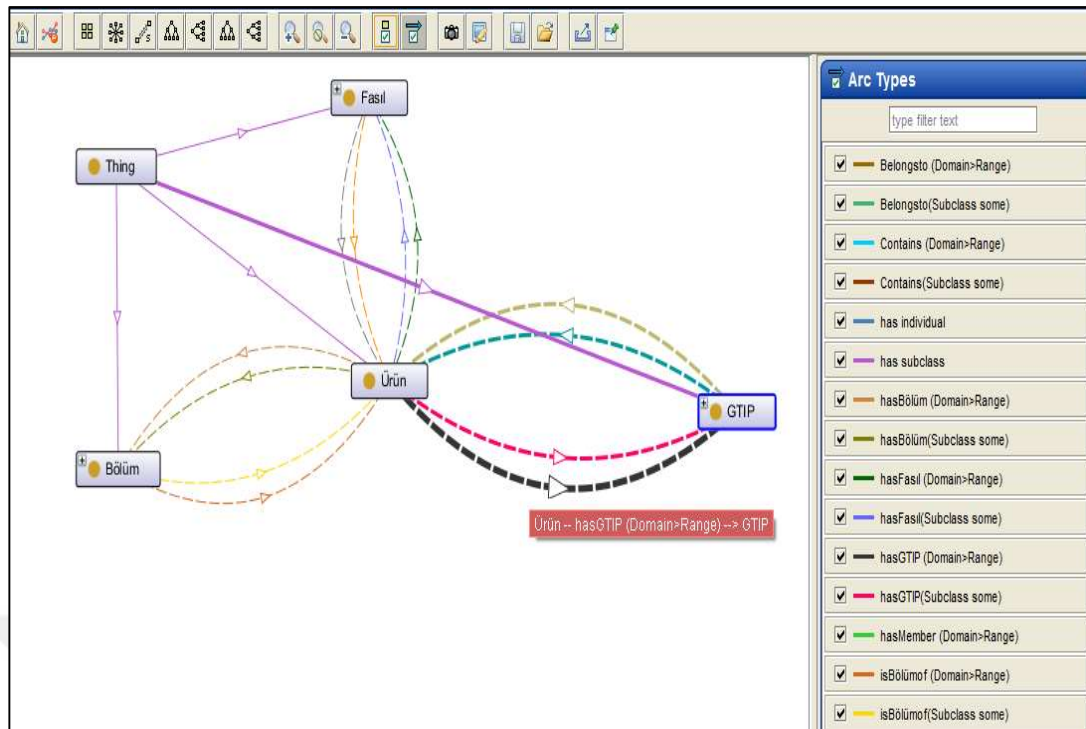


Figure 10. Example of Relations in the Protégé Ontology Editor.

Comparison of Ontology Tools

The even construction of ontologies depends on the domain itself and its complexity and ontology tools provide a graphical user interface and many features to ease the process of ontology construction (Khondoker & Mueller, 2010). There are number of different ontology construction tools. In broad perspective, the ontology tools can be compared according to:

- *General descriptions* in architectures, extensibility, storage and backup features,
- *Interoperability* features in import and export format,
- *Knowledge representation* features in KR paradigm, axiom language methodology,
- *Inference* capabilities as built-in and attached inference engines,
- *Usability* features in collaborative working capabilities, GUI and ontology libraries (Alatrish, 2013).

The comparison of ontology development tools is presented in Figure 11.

Features ↓	Apollo KMI(Open University)	Ontolingua KSL(Stanford University)	Ontosaurus ISI	WebOnto KMI	WebODE UPM	Protégé SMI(Stanford University)	OntoEdit Ontoprise	OILED University of Manchester
Pricing Policy	Open Source	Free Web Access	Open Source Evaluation	Free Web Access	Free Web Access Licenses	Open Source	Freeware and licenses	Freeware
S/w Architecture	Standalone	Client/ Server	Client/ Server	Client/ Server	3-tier	Standalone	Client/ server and standalone	Standalone
Extensibility	Plugins	None	None	No	via Plugins	via Plugins	via Plugins	No
Storage	Files	Files	Files	File	DBMS (JDBC)	File/ DBMS (JDBC)	File/ DBMS	File
Back up	×	×	×	√	√	×	×	×
Import Format	Apollo Meta-Language	Ontolingua IDL KIF	LOOMIDL ONTO KIF C++	OCML	XML, RDF(S), CARIN	XML,RDF(S), XML Schema	XML, RDF(S), FLogic, DAML+OIL	RDF(S), OIL, DAML+OIL
Export Format	OCML	KIF-3.0,CLIPS CML,LOOM,OKBC Syntax	LOOMIDL ONTO KIF C++	OCML, Ontolingua,GXL,RDF(S),OIL	XML,RDF(S),OIL, DAML+OIL,CARIN,FLogic,Prolog,Java,Java	XML,RDF(S), XML Schema, FLogic,CLIPS,Java,HTML	XML,RDF(S),FLogic, DAML+OIL,SQL-3	OIL,RDF(S),DAML+OIL, SHIQ, Dotty HTML
KR Paradigm	Frames	Frames+ FOL	DL (LOOM)	Frames+ FOL	Frames+ FOL	Frames+ FOL+ Meta Classes	Frames+ FOL	DL (DAML+OIL)
Axiom Language	Unrestricted	√	√	√	√	√	√	√
Methodology	×	×	×	×	√	×	√	×
Built in inference Engine	×	×	√	√	√	√	√	√
Attached inference Engine	×	ATP	√	×	Jess	JessFaCT FLogic	×	×
Automatic Classifications	×	×	√	×	×	×	×	√
GUI	√	√	×	√	√	√	×	×
Collaborative Working	×	√	√	√	√	×	√	×
Ontology Libraries	√	√	×	√	×	√	√	√

- √ ----denotes Yes
- × ----denotes No

Figure 11. Comparison of the ontology development tools (Singh & Anand, 2013).

2.1.4 Reasoners

Semantic applications are widely in use. Accordingly, reasoners play a key role in knowledge representation and reasoning capability of applications. (Bock et al., 2008) Reasoners check the existence of logical contradictions and consistency of the ontology. If the ontology is certain and consistent, (Abburu, 2012) they provide class hierarchy and reproduce information from the assertions of the defined ontology (Antoniou & Harmelen, 2004). As ontology sizes and their complexity grow in time, the need for logical reasoners increased accordingly (Tsarkov & Horrocks, 2006).

Description Logics (DL) reasoners are logic-based reasoners used for knowledge representation. DL Reasoners are used with many tools and API's. They support reasoning operations, for the capability of evaluating three main taxonomic errors of the ontology, which are inconsistency, incompleteness and redundancy (Gómez-Pérez, 1999)

The following inferences are evaluated by reasoners:

- Consistency of ontology,
- Class hierarchy,
- Object property hierarchy,
- Data property hierarchy,
- Class assertions,
- Object property assertions,
- Data property assertions,
- Same individuals,
- Different individuals,
- Disjoint classes.

There are a number of DL reasoners of OWL that can be used with Protégé. These are Pellet, RACER, FaCT++, Snorocket, HermiT, CEL, ELK, SWRL-IQ and TrOWL. The most popular reasoners are Pellet, FaCT++ and HermiT. Detailed descriptions of the popular reasoners will be presented in the following section.

FaCT++

FaCT++ (Fast Classification of Terminologies) is an open-source C++ based DL reasoner that supports OWL API, DIG and lisp-API, and has a tableau-based decision procedure for subsumption, satisfiability, classification and retrieval (“Old list of reasoners”, 2016). FaCT++ reasoner has a powerful ability in ontology reasoning on OWL ontologies that it firstly optimizes and preprocesses the representation of the ontology to determine inconsistency and entities, checks realizability of the rules and then it computes the classification of entities (Tsarkov et al., 2006).

Although FaCT++ determines the inconsistency in ontologies, it cannot present the reason of inconsistency. It can automatically determine possible inconsistency even in large scale ontologies.

FaCT++ performs two different practices in reasoning. Persistent reasoning practice is done by FaCT++ in three steps:

- Initialization of the reasoned,
- Application of the classification process,
- Recording of the gained information and internal state and modules them to be used in reasoning.

Most time-consuming part of this practice is the initialization process. At the second step, FaCT++ applies incremental reasoning. Rather than loading and classifying the ontology, it only collects information about a change in ontology. Moreover, it determines the variation of the ontology and computes these changes. These practices provide efficiency both for usage of resources and time in reasoning process. (Tsarkov, D. [Incremental and Persistent Reasoning in FaCT+], n.d.)

Pellet

Pellet is an open-source Java-based tableau reasoner available under license of MIT (“Old list of reasoners”, 2016). It supports OWL API, Jena and DIG, and provides subsumption, satisfiability, classification, retrieval and conjunctive query answering

of ontologies. Pellet has many features such as analyzing and repairing the ontologies, reasoning the data types, implication and querying. Pellet supports inconsistency checking and provides information about the reasons of inconsistency. Pellet also has a repairing process. Ontologies are parsed into RDF triples by RDF/XML parser, parsed ontology is processed for axiom specification and validation, and at this point, if it is needed, Pellet repairs the ontology. After specification process, axioms of classes are recorded into T-Box, and individual assertions are saved into A-box components. By using tableau approach, an XSD reasoner processes the reasoning. It is also compatible and efficient to use Pellet for developing ontology based applications that supports OWL API by its OWL API library or Jena by its Jena toolkit. Architecture of the Pellet reasoner is pictured in Figure 12. (Parsia & Sirin, 2004).

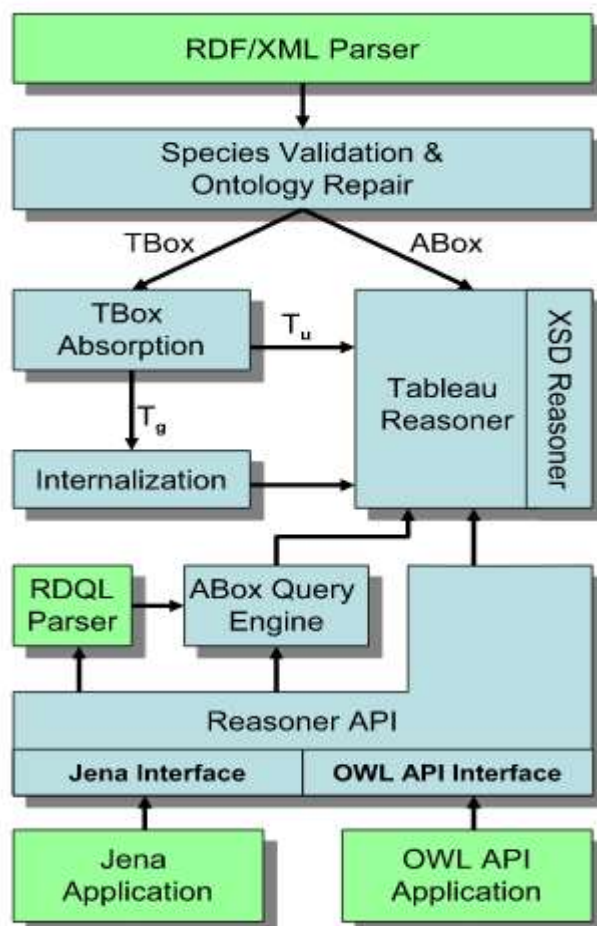


Figure 12. Illustration of Pellet architecture (Parsia & Sirin, 2004).

Hermit

Hermit is an open-source Java hypertext-based reasoner interface. Hermit is compatible with OWL API 3.0 and available under LGPL license (“Old list of reasoners”, 2016). Hermit uses all features of OWL 2.0 standards and provides many reasoning practices in classification and reasoning. Hermit supports SPARQL and DL-safe SWRL rules and graphs for representing the relationships. Hermit reasoner can be used through a Java interface for hypertext-based reasoning. In the interface a command line is used for basic reasoning. Hermit also provides the OWL API for integration to an application. Loading, classification and realization is performed by Hermit and a hypertext process takes place for reasoning. Reasoner interface converts OWL API data structures into its own data structure for the representation of ontologies. Furthermore, it also can convert its own data structure into OWL API data structure for ontology representation. Hermit also uses individual reuse and blocking practices for decreasing and optimizing the size of pre-models and detecting the cyclic blocks for improving its capability of consistency inspection. Architecture of Hermit is depicted in Figure 13 (Glimm et al., 2014).

In this thesis, Hermit, a popular and common tool, is used for consistency checking and reasoning processes with OWL API and Protégé. Relevant information is detailed in Chapter 5.

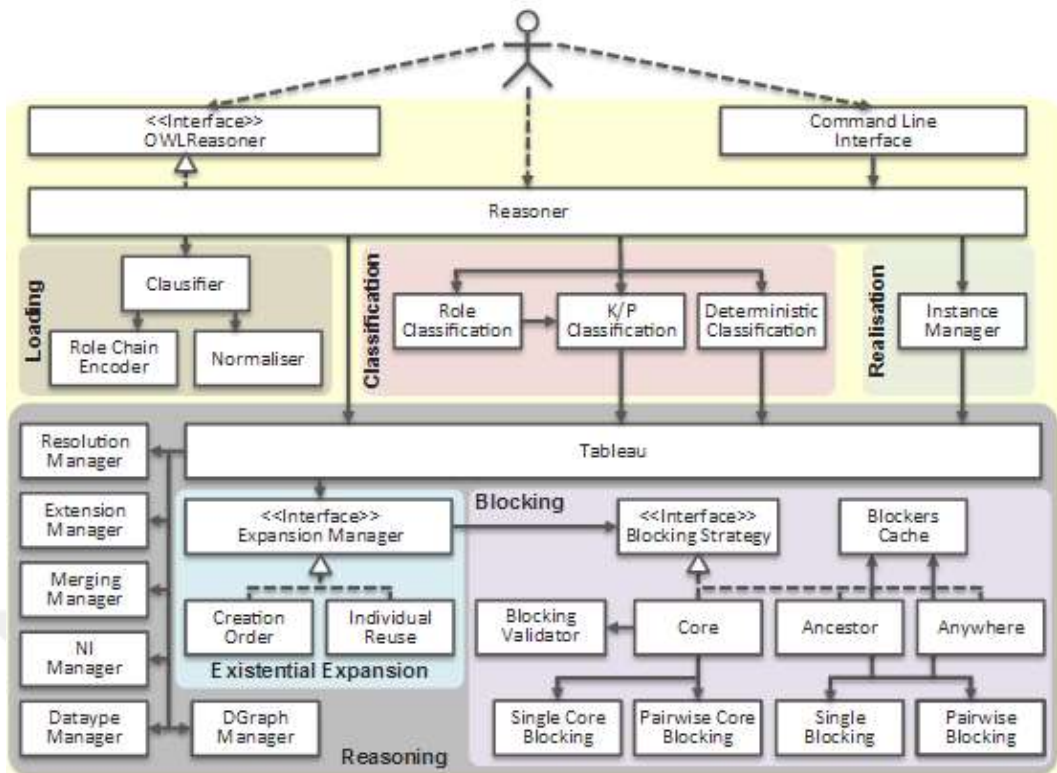


Figure 13. Illustration of Hermit architecture (Glimm et al., 2014).

Comparison of Reasoners

There are many different and popular methodologies and reasoners in semantic web community developed in many different structures and programming languages, having compatibility with different tools and APIs. Some of the reasoners and their reasoning capabilities and also features are discussed in the Figure 14.

	Pellet	RACER	FACT++	Snorocket	SWRL-TO	Hermit	CEL	TrOWL	ELK	
Methodology	Tableau based	Tableaux based	tableau based	Completion rules	SWRL rules	Hypertableau based	Completion rules	Completion rules	Consequence based	
Soundness	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Completeness	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	
Expressivity	SROIQ(D)	SHIQ	SROIQ(D)	EL+	-	SROIQ(D)	EL+	SROIQ	EL	
Native Profile	DL, EL	DL	DL	EL	-	DL	EL	DL, EL	EL	
Incremental Classification	Addition	Yes	No	No	Yes	Y/N	No	Yes	No	Yes
	Removal	Yes	No	No	No	Y/N	No	No	No	Yes
Rule Support	Yes (SWRL)	Yes (SWRL)	No	No	Yes (SWRL)	Yes (SWRL)	No	No	Yes (Own rule format)	
Platforms	all	all	all	all	all	all	Linux	all	all	
Justifications	Yes	Yes	No	No	Yes	No	Yes	No	No	
ABOX Reasoning	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	
OWL API	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	
OWL Link API	Yes	Yes	Yes	No	No	Yes	Yes	No	Y/N	
Protégé Support	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
NeOn Support	Yes	No	No	No	No	Yes	No	No	No	
License	DULI: AGPL	own	GLGPL	own	Y/N	GLGPL	Apache License 2.0	DULI: AGPL	Apache License 2.0	
Jena Support	Yes	No	No	No	No	No	No	Yes	Y/N	
Impl. Language	Java	LISP	C++	Java	Prolog	Java	LISP	Java	Java	
Availability	Open source	Commercial	Open Source	Commercial	Y/N	Open source	Open source	Commercial	Open source	

Figure 14. Comparison of Reasoners (Abburu, 2012).

As mentioned in the previous section, the most popular open source DL reasoners are Pellet, FaCT++ and Hermit. The following Figure shows the ontology classification times over several classified ontologies using OWL API. The charts show the number of ontologies at the vertical axis. The horizontal axis shows the time for different reasoners. As displayed in the chart, even though Hermit does not perform

better than other reasoners, it performs over complex ontologies systematically (Glimm et al., 2014).

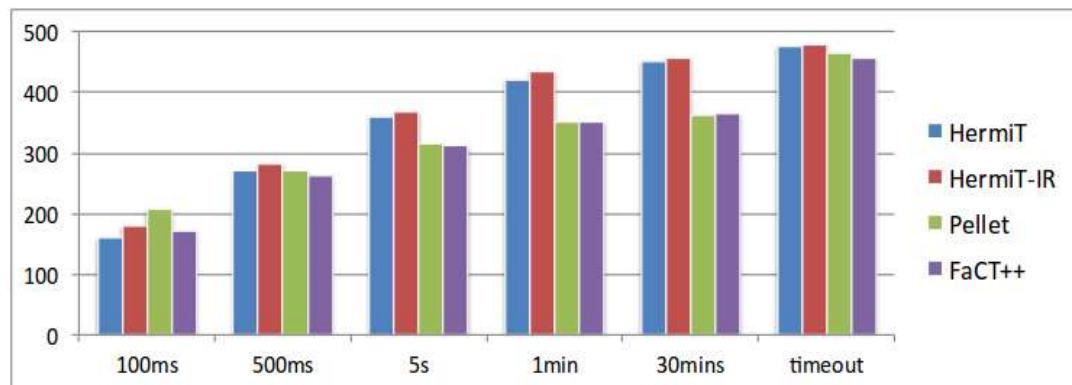


Figure 15. Reasoners comparison graphs (Glimm et al., 2014).

2.1.5 Knowledge Access

Ontologies can be accessed through APIs which are interfaces for the applications based on ontologies and used for loading, constructing, reconstructing, manipulating, saving and reasoning ontologies. The most popular open source APIs Jena and OWL API are explained in the following:

Jena

Jena API is a popular open source Java API based on the RDF platform. Jena models through RDF graphs which are interfaces to support RDF triples, and it also supports OWL. Jena is a language independent API, supporting many programming languages and an interface for ontology application development. Additionally, it also supports RDF/XML parsers and many reasoners for querying ontologies. Jena creates RDF models with the help of reasoners and injects these models into base RDF graphs. Jena is also capable of manipulating the RDF graphs. The components of Jena are illustrated in Figure 16 (“Jena Ontology API”, 2016).

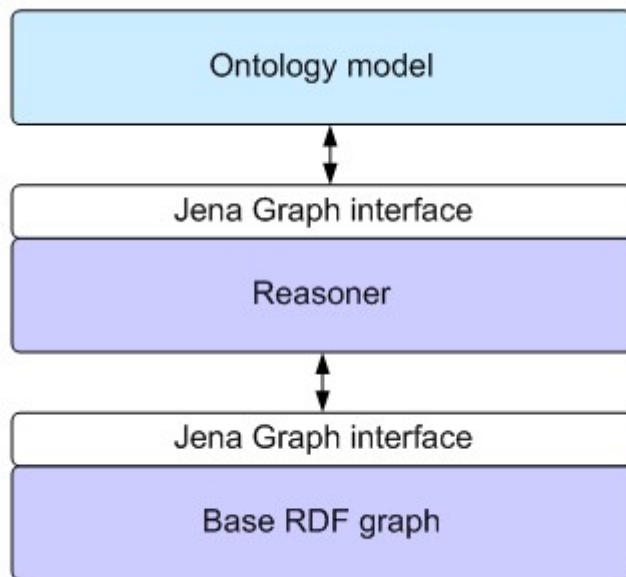


Figure 16. Jena components (“Jena Ontology API”, 2016).

Two main aims of Jena are providing a framework for ontology application development, and conformity to the statements of RDF. Jena supports both in statement and resource central approximations. Jena has many interfaces in accessing the resources, supporting connections between the resources, reasoning and representing the RDF graphs by its promising layered architecture (McBride, 2001).

Jena consists of three layers interacting with the interfaces which are called graphs for supporting RDF triples as a set of expressions. In Jena, Model layer is for accessing ontology applications through OWL, DAML+OIL and RDFS which support many features for resource accessing, viewing the graphs and graph nodes; Enhanced Layer stands for providing graph and graph nodes views, also provides polymorphism and inheritance of graphs; and Graph Layer works for providing an interface through the graphs. Java also provides a query language RDQL for the reasoning (Carroll et al., 2004). Jena architecture is presented in the following Figure:

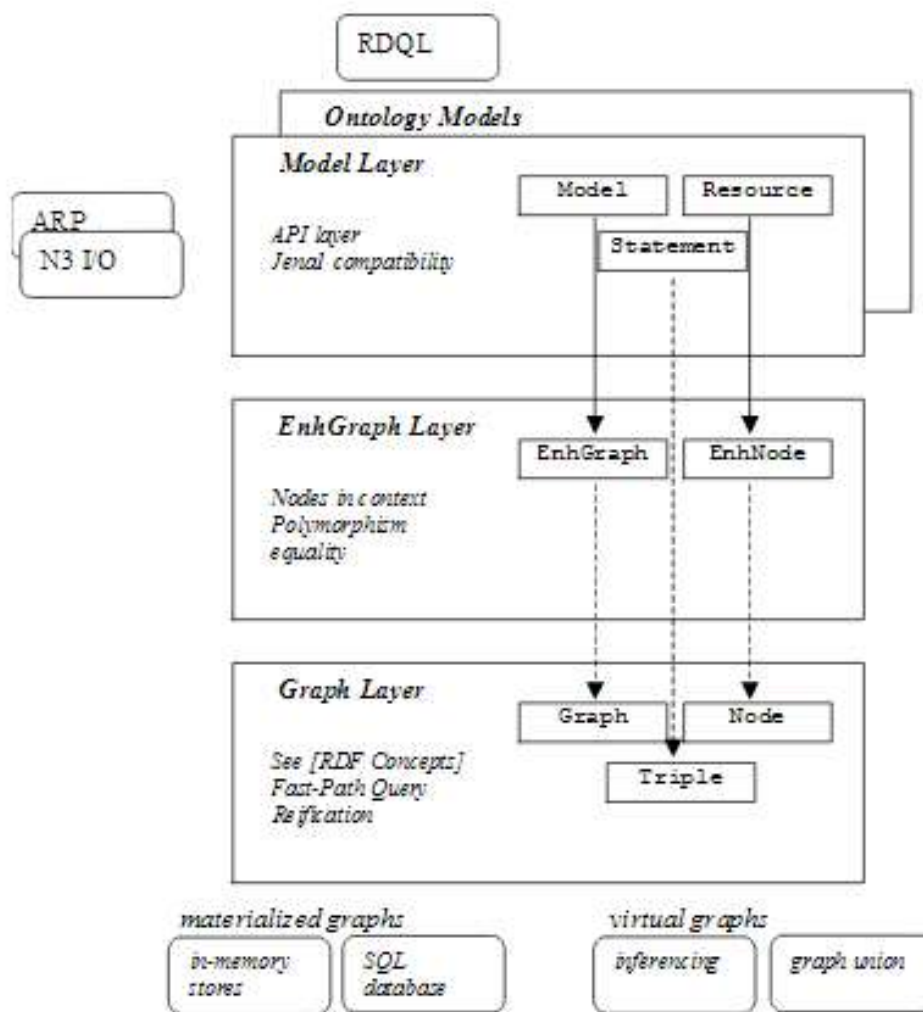


Figure 17. Illustration of Jena Layers (Carrol et al., 2004).

OWL API

The OWL API is an open source Java API that is focused on OWL 2 and is used for loading, constructing, reconstructing, manipulating, saving and reasoning ontologies. OWL API is available under either the LGPL or Apache Licenses. It is maintained by University of Manchester and is developed by the contributors as Clark & Parsia LLC and University of Ulm (Horridge & Bechhofer, 2009). OWL API supports manipulation of OWL ontologies but not in RDF level. OWL API is directly based on the OWL 2, as a set of axioms and annotations, and access to OWL ontology using interface for entities and class definitions. The names and hierarchies of

axioms, interfaces and class definition are similar to the specification structure (Horridge & Bechhofer, 2009). Design of OWL API is depicted in Figure 18.

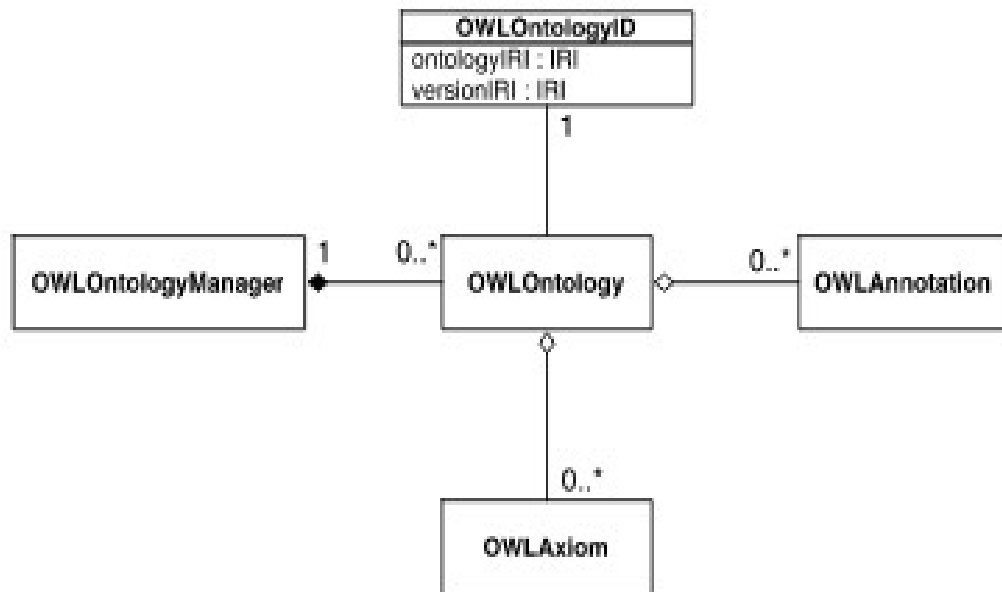


Figure 18. OWL API design illustration (Horridge & Bechhofer, 2009).

Due to the flexible design of OWL API, ontology management is done by Ontology Manager. Ontologies are created or loaded by OWLOntologyManager, which provides a centralized access and a centralized management on loading, customizing and monitoring of ontologies. Ontology change is done by OWLOntologyChange to change ontologies by adding, removing axioms and annotations through an ontology manager. Ontology parsing and rendering is done by OWL API parsers (Horridge & Bechhofer, 2009)

The OWL API uses the following components for parsing and rendering and supporting the OWL ontology using within applications:

- An API for OWL 2 and an efficient in-memory reference implementation,
- RDF/XML parser and writer,
- OWL/XML parser and writer,
- OWL Functional Syntax parser and writer,
- Turtle parser and writer,

- KRSS parser,
- OBO Flat file format parser,
- Reasoner interfaces (“The OWL API”, 2016).

OWL API uses reasoner interfaces to check inconsistency in ontology design, compute relations between entities, query ontologies and search for the extracted information from the ontology. OWL API uses the following reasoners:

- Chainsaw,
- FaCT++,
- JFact,
- HermiT,
- Pellet,
- RacerPro (“Reasoners, OWL API support”, 2014).

With its support on loading, constructing, reconstructing, manipulating, saving, reasoning and managing ontologies with its interfaces, OWL API is a cornerstone in ontology-based applications. In fact, OWL API is also popular between the developers of ontology-based applications because of its flexibility in design, supporting different types of implementations with its components (Horridge & Bechhofer, 2009).

There are many open-source resources such as codes and documentation about OWL API. Many ontology-based applications are using OWL API. A popular tool Protégé also uses the OWL API for ontology operation using the functionality of OWL API on ontologies (Horridge & Bechhofer, 2009). Protégé OWL API is an open-source Java library for OWL and RDF(S). Besides providing classes and methods for ontology operations, the API uses Description Logic engines for reasoning and also provides a graphical user interface by using user interface components of Protégé (“Protégé OWL API Programmers Guide”, 2010).

In this thesis, OWL API is used for ontology operations such as loading, manipulating and reasoning for the developed ontology application.

Structural Search

As studied before ontologies are the base structures for the semantic web. In order to make efficient reasoning, expressive querying is the main challenge in knowledge reasoning and still this issue is a hot problem in the field. The idea of the structural search is easy where the statement of the user is converted into an expressive representation language that can be compiled into a restricted language where this system provides a useful interface to user (Pan & Thomas, 2007). In Figure 19, this structure is illustrated:

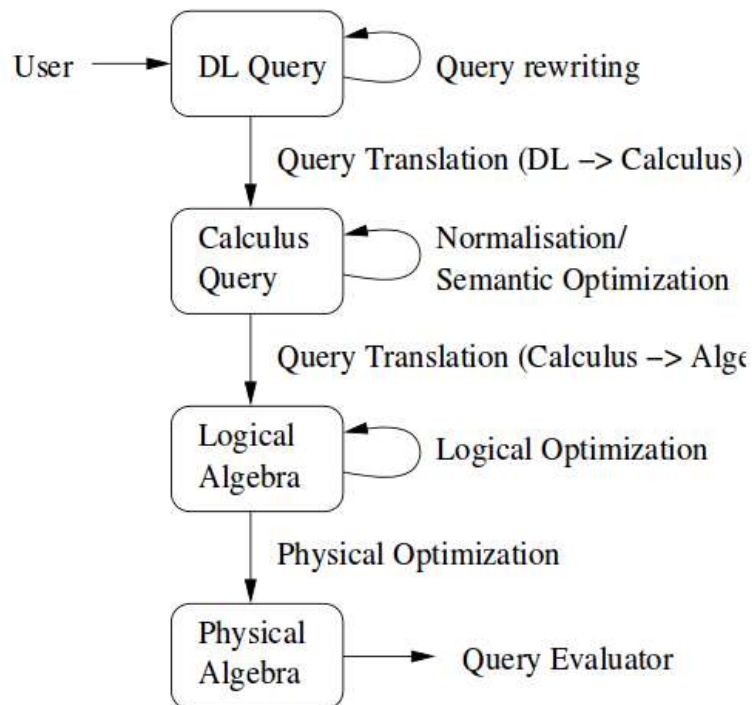


Figure 19. Illustration of DL Query (Peim et al., 2002).

SPARQL

SPARQL is a recommended querying system by W3C for RDF. SPARQL is used for bindings of RDF terms to querying from a set of RDF graphs. It is a simplified the query sentence as shown below:

SELECT variables

FROM data

WHERE {patterns}

It also provides filter expression such as union, disjunction, etc. (Euzenat et al., 2008).

SWRL

SWRL is used for rule interpretation on the Web. It is a combination of OWL DL and OWL Lite. SWRL provides to write horn-like rules that describe the OWL domain for reasoning about OWL individuals. These rules can be used for extracting new knowledge from the existing knowledge base. SWRL specification provides designers to use a number of rule engines freely to reason with the SWRL rules that recorded in an OWL knowledge base (O'connor et al., 2005).

DL-Query

As mentioned in the previous section, DL query has high level of expressivity. Nevertheless, it brings some restrictions with respect to this expressivity such as in order to use DL query, designer must be aware of these restrictions (Zhang & Miller, 2005).

2.2 Background on Natural Language Processing

Natural Language Processing (NLP) is a field of computer science, artificial intelligence (AI) and linguistics. Main concern of the branch is to develop an interaction between computers and humans using natural language (Reshamvala et al., 2013).

2.2.1 Natural Language Processing (NLP)

As a standard definition, NLP is a computer program which understands human language in written or spoken form. NLP is an approach for analyzing text where the analysis is based on the language rules such as the syntactic and morphological

structures. NLP is a multidisciplinary area that covers computer and information science, mathematics, linguistic, psychology and several engineering fields (Chowdhury, 2003).

NLP takes place in many researches for different purposes. Some of the research areas that NLP is used are:

- Machine translation,
- Morphological analysis,
- Automatic summarization,
- Question answering,
- Information extraction,
- Information retrieval,
- Classifiers,
- Text similarity (“Introduction to Natural Language Processing”, 2015).

NLP also takes place in Speech Synthesis where text to speech transformation and speech recognition works for simultaneous transformation from speech to semantic information using grammatical rules and syntactical analysis of the related language (Reshamvala et al., 2013).

In literature, there are a number NLP tools and algorithms developed for different purposes. These tools, named NLP software, can be classified as, general information tools, taggers and morphological analyzers, information retrieval and filtering tools, machine learning tools, finite state automata tools, Hidden Markov Model tools, language modeling tools, corpus tools, etc (Liddy et al., 2000).

Levels of NLP

NLP has to analyze the given sentence on different levels, such as semantic, morphological, lexical, etc. and to reach the content of the text. In addition to these, NLP is also directly related to computational linguistics which is mainly related to Cognitive Science and is seen as a plausible way to understand the human cognitive

system to model language usage; and a way to understand cognitive processes of the human mind (Liddy, 2001).

In the notion of language processing, there are multiple levels, such as semantic, morphological, lexical, etc., in analyzing any natural language. It is commonly known that all these levels are taken into account when an NLP system is to be designed. However, there are various NLP systems where some systems analyze all of these levels; on the other hand some only use one of these or a combination of levels. As a result, the only difference between NLP systems is whether the system uses weak NLP or strong NLP (Liddy, 2001).

As a rule of natural language of humans, every level of natural language consists of some information used from extracting meaningful knowledge from the natural language. The deeper level of language used in NLP means, the more capable system can be designed (Liddy, 2001). At the center of any NLP, main issue is the understanding of the natural language and the core task of the NLP programs is to solve three main challenges. The first is the thought (meaning of a set of input), the second is meaning of input and its representation and the last one is the world knowledge. In this context, inevitably, all NLP system must have different levels to solve the mentioned issues. The levels of NLP are presented below (Chowdhury, 2003):

Phonology consists of meaning about the sound of each word.

Morphology is the multicomponent structure of the input word. Morphology provides convenient and practical information for parsing and finding lemmas (Daille et al., 2002). As general information, words can be in root form, or can include some other suffixes. For instance, *computer* can be divided into two parts as *compute* and *er* where the first part *compute* is the root of the word, and *er* is the suffix of the word. The small units of the word are mentioned as morphemes, *compute* and *er*. As a result, a given word conveys other meanings when compared to the sum of its parts. Another example in Turkish can be seen in Figure 20.

	1 st morpheme	2 nd morpheme	3 rd morpheme	4 th morpheme
word 1	hasta	-lan	-dı	-m
hastalandım				
word 2	yaş	-lan	-dı	-m
yaşlandım				

Figure 20. Morphological analysis of words in Turkish (Yavuz, & Balpınar, 2011).

Word 1: *Hastalandım* meaning *I got sick*, and word 2 *yaşlandım* meaning *I grew old*.

Another Turkish word *ölümsüzleştiriveremeyebileceklerimizdenmişsinizcesine* can break down into morphemes as follows (Sak et al., 2011):

ölüm+süz+leş+tir+iver+eme+yebil+ecek+ler+imiz+den+miş+siniz+cesine.

Intuitively, each morpheme is recognized with human cognition and its meaning is understood. Similarly, the NLP system needs to find all morphemes and extract information about the words from these morphemes. At this point, availability of a computational tool for morphological analysis of any language may be the first building block of any higher order NLP application (Pretorius & Bosch, 2003).

Lexical means interpretation of each word's meaning. This meaning representation can differ according to the semantic theory of the developed NLP system. On the other hand, in the light of the generative grammar and structuralism, it is stated that lexical meaning is a starting point of any semantic theory (Katz & Fodor, 1963). Therefore, the core paradigm is the decomposition of all lexicons into basic logical primitives. For example, *apple* can be given in such logical form (Table 2).

Table 2. Logical representation of *apple*.

Apple (an organic entity that can be eaten by human, grows on tree, has color, has round shape)
((CLASS FRUIT) (PROPERTIES(COLOR(RED,YELLOW,GREEN)))
(USAGE(PREDICATION(CLASS(FOOD)(OBJECT HUMAN))))

As seen in the given table above, finding a finite set of primitives for a large number of lexicons is the central difficulty of the theory. On the other hand, recognizing the verb functionality of the verbs and their thematic roles is the most influential part of the theory (Ovchinnikova, 2012). Furthermore, in the framework of Pustejovsky (1991), there are additional challenges with respect to meanings of lexical items. First, there is no way to understand the meaning of word, if the syntactic structure of the given sentences is not taken into consideration. Even if the syntactic structure is taken into consideration, the biggest problem is, each lexical item has its own deeper meaning that may change its own meaning according to the system and the domain it operates in (Pustejovsky, 1991). Consequently, lexical analysis cannot be done without syntactic analysis. Therefore, it is necessary for any NLP tool that lexical analysis must be done in light of the syntactic analysis of a given sentence.

Syntactic analysis, is analyzing the words in the sentence for extracting information about the dependency and relationship of the words in the given sentence. Therefore, any NLP system has to have information about the grammar of the related language and appropriate parsing of the sentences. For example, the following sentences have different syntactic structures. You can see two different structures in Figures 21 and 22.

Sentence1: “Kim wrote that book with the blue cover”

[_{NP}Kim [_{VP}wrote [_{NP}that book with the blue cover]]]

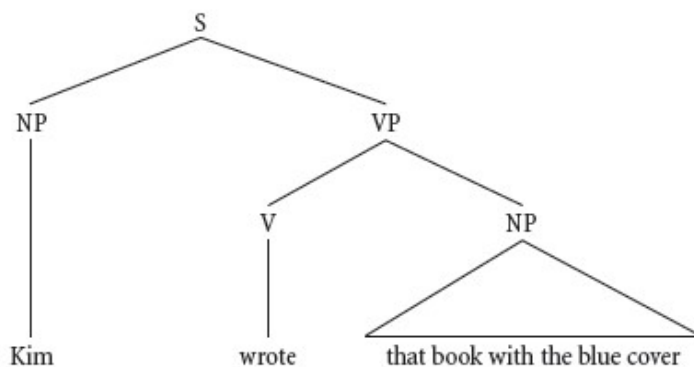


Figure 21. Syntactic structure of Sentence1 (Tallerman, 2014).

Sentence 2: “Kim bought that book with her first wages”

[_{NP}Kim [_{VP}bought [_{NP}that book][_{PP}with her first wages]]]

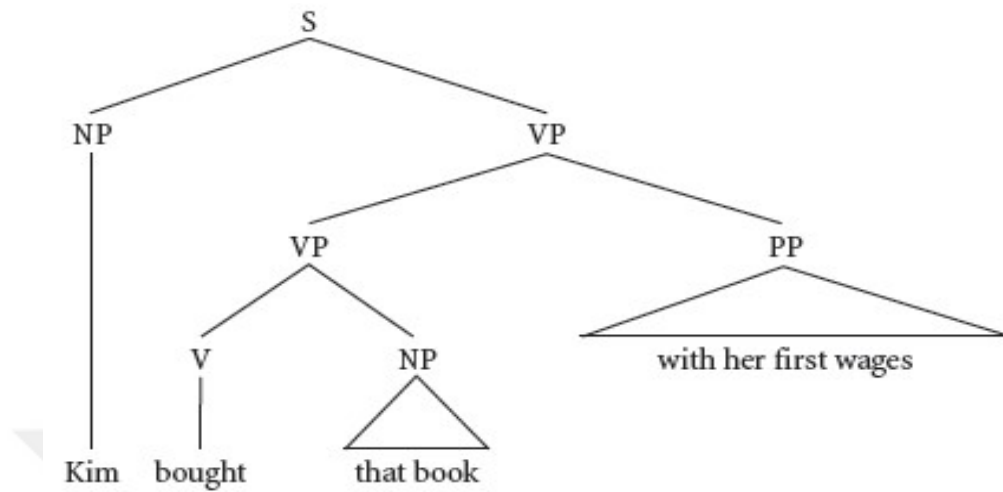


Figure 22. Syntactic structure of Sentence2 (Tallerman, 2014).

There are numbers of rules which identify the syntactic structure of the sentences. In the above sentences, different syntactic structures can be seen in Figure 7 and 8. Yet, sometimes syntactic analysis is difficult due to ambiguity. For example:

Sentence 3: I saw a man with a binocular

The above example is famous since is impossible to resolve the ambiguity. The ambiguity of the given sentence can be defined as, “I saw a man who has a binocular” vs. “I saw a man with a binocular”. Nevertheless, using an analysis of discourse or context can solve the ambiguity. As a result, syntactic analysis of the sentence is the only way to get the difference between the sentences even if they have the same words. However, using an additional level of NLP analysis is necessary in ambiguous situations.

Semantic analysis describes the meaning of the sentence by detecting the relations of the words of the sentence with each other. The semantic process is necessary for pragmatics level in providing the meaning of the sentence and it also has an essential

role in the disambiguation of the words by detecting the meaning from the whole sentence.

Discourse is another level of an NLP system that stands for dealing with a unit of text which is longer than a sentence. Discourse of the sentences in the text determines the function of the sentences in the given text. For example, in a regular magazine text, discourse elements can be classified as identification of the sentences' function such as, main story, evaluation, attribute, and expectation of the text. In order to reach the correct interpretation of the meaning, it is necessary to make discourse analysis of the given clauses in the text. In short, the most vivid explanation can be given in a short discourse such as:

Sentence 4: Mary and John only go to cinema when a Turkish movie is playing.

In the Sentence 4 discourse, we infer the conditional relation between two different events. Even if there are number of interpretations for a continued event, with the help of the discourse analysis there is further information can be extracted (Gardent & Webber, 1998) such as:

Sentence 5: John and Mary go to cinema rarely.

As a result, discourse analysis stands for the relation of clauses rather than the smaller unit of a language.

The last level of the NLP system is *pragmatic*. Analyzing the context dependency of the text, usage of this level is especially important for solving the disambiguation where similar morphemes may convey different meanings. For example, the Turkish word *yüz* can convey meaning of swimming activity but also has a different meaning such as the number hundred. These different meaning can be extracted with respect to context of the text (Liddy, 2001).

In light of the information above, all these levels are not necessary for every NLP system; using appropriate level or levels can improve the effectiveness of the system. Since rule governed levels are more structured and efficient to use in limited domain,

it is claimed that the lower level of analysis are effective for dealing with the smaller units of analysis (Liddy, 2001).

Approaches of NLP

There are typically three main approaches developed by researchers with respect to NLP systems.

Connectionist approach is one of these approaches where connectionist networks learn from experience rather than the predetermined rules. Here, large data set are used which includes a number of examples from the target language. The connectionist NLP system can be classified as a system which can learn from previous experiences (Reilly & Sharkey, 1992). From a general perspective, language is taken into account as a product of cognitive processes. The main aim of this school is to find evidence that not only the form but also the meaning of any expression cannot be enough to explain the knowledge of a person. In this school, all the mental models of the person, the person's ability such as projection of the physical model to abstract model, usage of super-positional representation and constraints satisfaction schemes of the person are, taken into account and integrated as a multiple source of information (Sharkey, 1992).

Statistical approach uses mathematical techniques in order to extract information from a large corpora where statistical methods are used in order to develop a generalized model of the linguistic phenomena, which is based on instances of the phenomena, that is taken from the processed corpora without any addition about linguistic knowledge (Liddy, 2001). Statistical NLP can be defined as a quantitative approach to automated language processing where quantitative means using probabilistic modeling, information theory and linear algebra (Manning & Schütze, 1999).

Symbolic approach is the last approach which is used in the developed system. Symbolic approach is a type of top-down approach which extracts information from text using known grammatical patterns and meaning of associations of the text (Jackson & Moulinier, 2007). Without any doubt, this type of NLP is one of the most

obvious NLP approaches that dominated by 20th century structuralism. This approach considers the center of the study as a matter of strings where each text is actually a sequence of sentence, each sentence is a sequence of words, and each word can be decomposed to a letters sequence. In this aspect, each contributor has to be processed; continuous information can be predicted with the existence of what is already known. And in second place, symbolic approach works at least in several ways where the analysis is not performed on the whole text but can be managed as a set of strings (Laporte, 2005).

2.2.2 Turkish NLP

Turkish is one of the most common languages all round the world with its variety of dialects, and accents. Turkish is an extremely different language compared to Indo-European languages. The first reason of this difference comes from the morphological structure of Turkish. Turkish is an agglutinative language, therefore, in Turkish, inflectional and derivational word alternation can be done by suffixes. Furthermore, in a sentence, relations between words are given by suffixing (Carki et al., 2000). Although it is a common language, there are a limited number of researches that has been done in Turkish Natural Language Processing (Orhan et al., 2011). Many academic researches have been done for the analysis of Turkish but few commercial tools exist. Some of the implementations, dictionaries and morphological analysis tools presented in the academic research are given below:

- An online Turkish dictionary provided by Turkish Language Foundation (Orhan et al., 2011),
- Turkish WordNET, Turkish common sense database (Amasyalı, 2012),
- Implementation of two-level morphological description of Turkish word structures (Ofłazer, 1994),
- An affix stripping morphological analyzer for Turkish (Eryiğit & Adalı, 2004),
- Zemberek, An open source NLP framework for Turkic languages (Akın & Akın, 2007; “Zemberek Contributors”, 2013),

- TRmorph, A freely available morphological analyzer for Turkish (Çöltekin, 2010),
- Two-level Turkish morphological analyzer (Şahin et al., 2013),
- ITUmorph, A Turkish morphological analyzer (Şahin, 2014),
- Implementations of a morphological parser, morphological disambiguator and web corpus as Turkish language resources. (Sak et al., 2008),
- ITU Turkish NLP Web Service, A Turkish NLP platform, (ITU) (“ITU Turkish NLP”, 2015); (Eryiğit, 2014).

Some of these tools will be explained below.

TRMorph

TRMorph is a Turkish morphological analyzer that can analyze the morphemes of a given input text based on rules. As a general approach, TRMorph uses finite state transducers (FSTs). It is stated that this tool was initiated for reducing morphological analysis' time cost, as the nonexistence of free Turkish morphological analysis tool opportunity (Çöltekin, 2010).

Turkish WordNet

WordNet is a project where the system provides an efficient integration of traditional lexicographic information and modern computing. Basically, it is an on-line database controlled by a program where the database consists of verbs, adjectives, nouns and adverbs organized and defined as a set of synonyms that stand for lexicalized concepts. Each semantic relation is connected to a synonym set (Miller, 1995). Turkish WordNet is a project which is actually connected to WordNet online database. There are six different languages that connect to the WordNet, and Turkish WordNet is one of these languages. Turkish branch of the WordNet is participated with the Sabancı University, The Human Language and Speech Technologies Laboratory within the participation of Balkanet Project (Bilgin et al., 2004).

Zemberek

Zemberek is the most reliable and popular Turkish NLP tool that is an open source code where a number of contributors can access and develop the system (“Zemberek Contributors”, 2013). Zemberek provides spell checking, morphological analysis, word derivation, word suggestion, word construction and translation of the special Turkish characters into English form using ASCII similarities, and spelling of the given text (Akın & Akın, 2007).

Zemberek provides a class of lexemes such as verb, noun, adjective, and reduces the word into smaller units such as root and affixes. Zemberek is chosen as the symbolic analysis tool for the developed application.

The library of Zemberek consists of a core and some special language structure information. Core has been developed especially for Turkish language; however, it is useful for other languages; there is no need to change the coding in the core structure of the system. All language implementation can be done from the user interface (Akın & Akın, 2007).

A morphological analysis example using Zemberek for the Turkish word “ölümsüzleştirmek” is illustrated in Figure 23:



Figure 23. A morphological analysis example using Zemberek.

2.2.3 Ontology Construction Using NLP Techniques

Ontologies have a common and an extensive role in knowledge representation. There exist many theories on ontology construction, ontology reuse, ontology mapping, ontology learning, etc. Definition and creation process of a knowledge base is named ontology population and automation of ontology construction, is named ontology learning (Buitelaar et al., 2005). Many engineers and researchers are still trying to develop systems which learn from texts, focused on ontology learning theories and techniques as a study of NLP, and most are still trying to extract information to ontologies by using automated ontology learning powered with NLP techniques. On the other hand, there are many research and engineering efforts in developing new ontology construction techniques and processes using NLP.

Using NLP while constructing an ontology is a known procedure where there are a number of examples such as, Copestake and her friends (2006) built a semi-automatically extending ontology in the chemistry area using NLP techniques for scientific texts.

The difficulty in ontology construction arises because of the diversity in the description of the domain from many different perspectives, domain data and techniques which force the constructors to make a construction decision from this diversity. The difficulty may be mostly overcome by constructing the ontologies according to purpose of the ontology, data relevant to ontology construction and users (Jaimes & Smith, 2003).

To have a complete and reliable ontology construction, there are three ways to construct ontologies (Subhashini & Akilandeswari, 2011).

Manual ontology construction requires domain expertise, time and much effort. It is reliable but difficult to construct.

Automatic ontology construction is the most common way of constructing ontologies for time and money reasons. Statistical approaches are used in learning from domain texts, corpus, documents and extracting the relevant information and relationships to

ontologies. The more effort in automatic ontology construction is required in selecting the necessary concepts and relationships.

Semi-automatic ontologies constructor extracts the information with human intervention to have a complete and reliable ontology (Jaimes & Smith, 2003).

The aim of the literature review was to identify potential concepts and perspectives about ontologies and NLP. In the end, the reviewed articles presented clear sense about how to build an ontology with an NLP tool. To sum up, review of the literature can be separated into two parts. The first part provides general concepts of ontologies and different types of ontology construction methods. The second part gives broad information about natural language processing and particularly Turkish NLP tools. Theoretical framework of the developed application is explained in Chapter 4.

CHAPTER 3

ANALYSIS AND DESIGN OF THE SYSTEM

As mentioned in Chapter 2, ontology construction for a domain can be done from many different perspectives, according to the domain data and techniques used for ontology construction. Here, the main point is making a decision by focusing on the purpose of the ontology by considering the domain data for a descriptive and meaningful ontology construction.

Consequently, in this thesis, to decide on the ontology design, the domain data is analyzed first. Secondly, according to domain data analysis; entities, relations, data properties and NLP techniques are also decided and the ontology is constructed. After the ontology construction, a system is developed which inputs user product description sentence and outputs the searched G.T.I.P. number by a GUI. The steps followed and the implementation details of analysis and design of the system are presented in the following sections:

3.1 Ontology Design

The declared goals of the thesis are the analysis of domain data, modeling and construction of the product ontology and testing of the constructed ontology. The following sections present detailed information about all the related work for a reliable and a useful ontology design with the help of Turkish NLP.

3.1.1 Analysis of Domain Data

Analyzing the domain data is essential for ontology construction. Classes, subclasses, entities and their relations, objects and data properties cannot be specified without a detailed domain data analysis. At this point, detailed information about our domain, namely, HS coding system, is provided.

HS coding system currently consists of 5000 groups of products each described by six-digit codes that are similar in all countries, then an additional six-digits, refers to a particular product from the referred group that varies between countries. In total, it is a twelve-digit HS code.

HS codes are specified according to 21 main sections called *Bölüm* in Turkish Foreign Trade. Each main section has chapters called *Fasıl* and each chapter has product groups and products specified by different HS codes for each category of product.

In this study, a part of the current HS coding system, namely Section 11 is chosen as the subdomain of our ontology. This main section is defined within the HS coding system as “Dokumaya Elverişli Maddeler ve Bunlardan Mamul Eşya”. One of the chapters of Section 11, namely Chapter 50 which is defined as “İpek” is chosen as point of interest; and Chapter 50’s position in HS coding structure is presented in the following Figure:

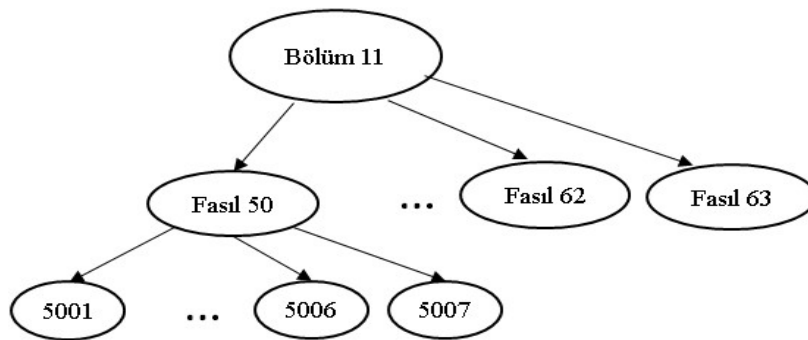


Figure 24. Position of Chapter 50 in HS coding system.

The general structure of the HS coding subsystem is given in Tables 3, 4 and 5.

Table 3. Chapter list of the Section 11

(Dokumaya Elverişli Maddeler ve Bunlardan Mamul Eşya).

Fasıl No	Bölüm 11 - Dokumaya Elverişli Maddeler ve Bunlardan Mamul Eşya
50	<i>İpek</i>
51	yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat
52	Pamuk
53	dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat
54	sentetik ve suni filamentler
55	sentetik ve suni devamsız lifler
56	vatka, keçe ve dokunmamış mensucat; özel iplikler; sicim, kordon, ip, halat ve bunlardan mamul eşya
57	halılar ve dokumaya elverişli maddelerden diğer yer kaplamaları
58	özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler
59	emdirilmiş, sıvanmış, kaplanmış veya lamine edilmiş dokumaya elverişli maddelerden mensucat, dokumaya elverişli maddelerden teknik eşya
60	örme mensucat
61	örme giyim eşyası ve aksesuarı
62	örülmemiş giyim eşyası ve aksesuarı
63	dokumaya elverişli maddelerden diğer hazır eşya; takımlar; kullanılmış giyim eşyası ve dokumaya elverişli maddelerden kullanılmış eşya; paçavralar

Table 4. Product Groups list of Chapter 50 (İpek).

G.T.İ.P.	FASIL 50-ipek
5001	Çekilmeye elverişli ipek böceği kozaları
5002	Ham ipek (bükülmemiş)
5003	İpek döküntüleri (çekilmeye elverişli olmayan kozalar, iplik döküntüleri ve ditme suretiyle elde edilen döküntüler dahil)
5004	İpek ipliği (ipek döküntülerinden elde edilen iplikler hariç) (perakende satılacak hale getirilmemiş)
5005	İpek döküntülerinden elde edilen iplikler (perakende satılacak hale getirilmemiş)
5006	İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş iplikler (perakende satılacak hale getirilmiş); misina (ipek böceği guddesinden elde edilen)
5007	<i>İpek veya ipek döküntülerinden dokunmuş mensucat</i>

Table 5. Product list of product group 5007

(İpekveyaipekdöküntülerindendokunmuşmensucat).

G.T.İ.P.	5007 – Açıklama	Seviye
5007	İpek veya ipek döküntülerinden dokunmuş mensucat	0
5007.10	Buretten mensucat	1
5007.10.00.00.11	El tezgahlarında dokunanlar	2
5007.10.00.00.19	Diğerleri	2
5007.20	Diğer mensucat (ağırlık itibariyle % 85 veya daha fazla ipek veya ipek döküntüsü içerenler) (buret hariç)	1
	Krepler	2
5007.20.11.00.00	Ağartılmamış, temizlenmiş veya ağartılmış	3
5007.20.19.00.00	Diğerleri	3
	Saf ipekten ponje, habutai, honan, şantuk, korah ve benzeri diğer Uzak Doğu kumaşları (buret ve diğer ipek döküntüleri veya dokumaya elverişli maddeler karıştırılmamış)	2
5007.20.21.00.00	Bez ayağı (düz dokunmuş) (ağartılmamış veya temizlemeden daha ileri bir işlem görmemiş)	3
	Diğerleri	3
5007.20.31.00.00	Bez ayağı (düz dokunmuş)	4
5007.20.39.00.00	Diğerleri	4
	Diğerleri	2
5007.20.41.00.00	Seyrek dokunmuş	3
	Diğerleri	3
5007.20.51.00.00	Ağartılmamış, temizlenmiş veya ağartılmış	4
5007.20.59.00.00	Boyanmış	4
	Farklı renkteki ipliklerden	4
5007.20.61.00.00	Genişliği 57 cm. yi geçen fakat 75 cm. yi geçmeyenler	5
5007.20.69.00.00	Diğerleri	5
5007.20.71.00.00	Baskılı	4
5007.90	Diğer mensucat	1
5007.90.10.00.00	Ağartılmamış, temizlenmiş veya ağartılmış	2
5007.90.30.00.00	Boyanmış	2
5007.90.50.00.00	Farklı renkteki ipliklerden	2
5007.90.90.00.00	Baskılı	2

In Turkish Foreign Trade product structure, all sections are classified according to their chapters and are different from each other such that their chapters and products are not related to any other section. This means, there is no relation between chapters, and also no relation between products of different chapters.

In the HS coding system each product has its own *G.T.İ.P.* (HS code), *Açıklama* (explanation of product) and *Seviye* (to describe the sub-class level of product in class structure) data. In the detailed analysis of product groups and their products, as listed in Table 4, some of the products do not have *G.T.İ.P.* numbers (e.g. *Krepler*) but they have *Seviye* data. *Seviye* data enforces the creation of class structure according to *Seviye* value and this is a challenging issue. Also, some of the products have same or common names (e.g. *Diğerleri*). In other words, the product “Diğerleri” can be a member of any chapter or any class.

The result of the detailed analysis of the domain data implies a systematic class hierarchy that needs to be kept as it is already inherent in HS coding system but additional classes and relations are needed in the ontology model. Furthermore, each class of the ontology has a unique name and these names need to be represented in our ontology. Using keywords (*AnahtarKelime*) which are descriptive words of the related product appeared as a possible solution. It is thought that producing keywords using NLP techniques is a necessary operation to improve the querying quality and increase the accuracy of the system. Note that, in order to find an HS code, user will query the system with a product's description where the description is done by the users' daily language. Therefore, related keywords are specified using NLP techniques.

As a result; analyzing the name and related information of products with an NLP tool will systematize our keyword production which will increase the reliability of our novel approach.

3.1.2 Modeling and Construction of Ontology

Since our domain is wide, and it almost defines all the things (living animals, textile, etc.) it is difficult to design an ontology using regular aspects. Note that, *silk* can be used to describe animals, textiles, adjectives and specifications. Therefore, instead of using regular relational definitions, different approaches such as using class restrictions and data property descriptions need to be used together for relating objects.

In this study, decisions about ontology model and specifications of ontology structure, classes, entities, relations, data properties and construction of the relations are detailed in the following sections.

Specifying the Ontology Structure

As mentioned in the previous chapters, in Turkish Foreign Trade products structure, all sections are different from each other such that their chapters and products are not related to any other section. This means that, there is no relation between sections. So in the constructed ontology, each section is defined as a class. And these classes are given the name used in the HS coding system which is also important for traceability of the classes in the ontology. In the ontology, chapters of each section are defined as sub-classes of their sections meaning there is no relation between chapters.

The challenging work in defining the product groups and their products is, as listed in Table 4, that some of the products do not have G.T.İ.P. but they have a *Seviye* data (e.g. *Krepler*: no G.T.İ.P, *Seviye:2*). In addition, some of the product names are the same (e.g. *Diğerleri*) as different products of different chapters. According to all these restrictions, products of the chapters are defined as sub-classes of their chapters by considering their *Seviye* data and G.T.İ.P. numbering structure in the constructed ontology. As an example, the class hierarchy graphs of *ipek* class are illustrated in Figure 25 and 26; and class hierarchy of the ontology from Protégé defined in Figure 27.

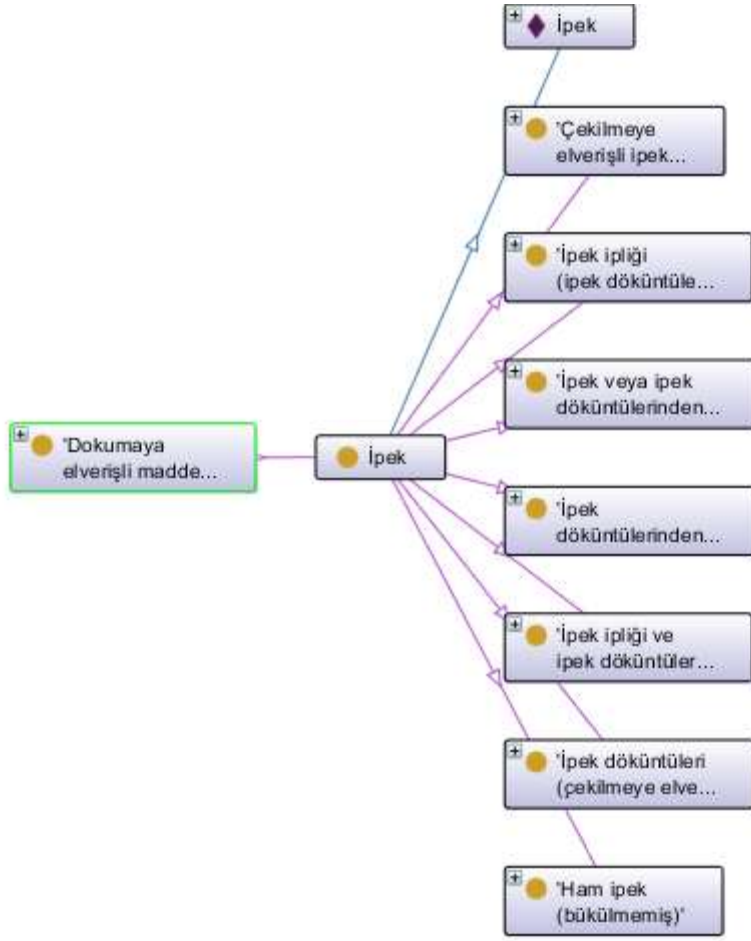


Figure 25. Class hierarchy graph of *İpek* class.

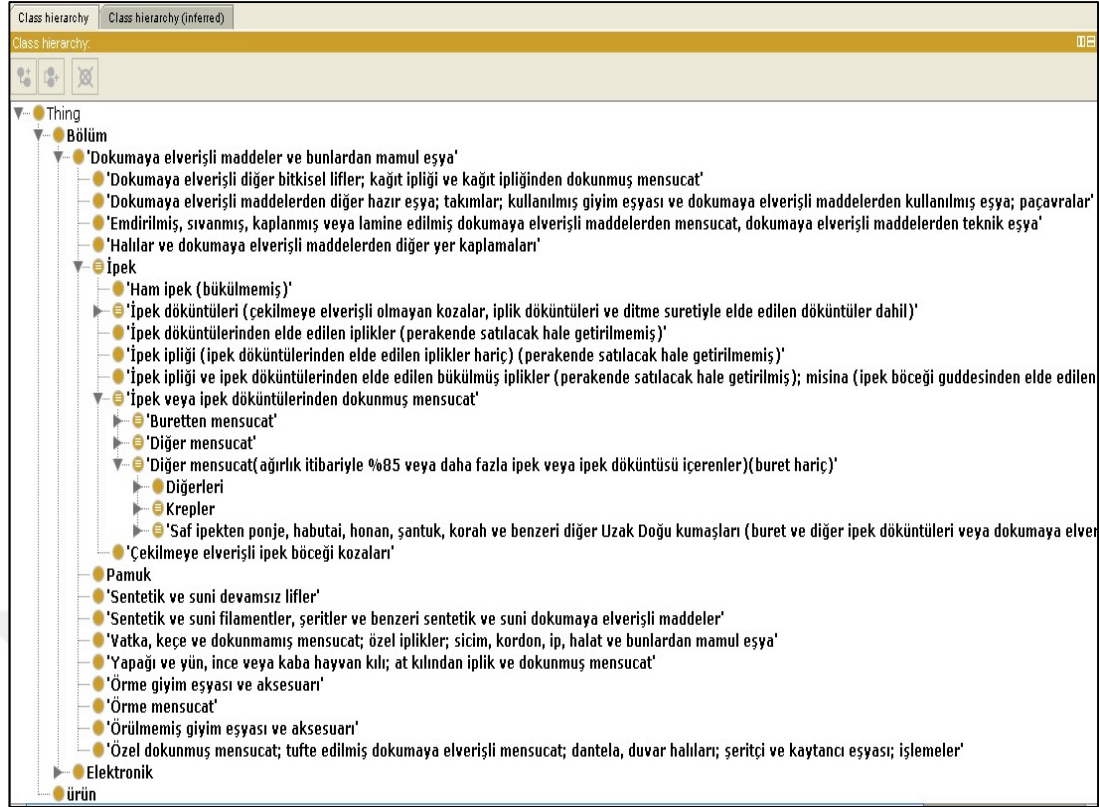


Figure 27. Class hierarchy of Ontology from Protégé.

As shown in Figure 27 above, there is also another class defined, named, *Ürün*. The need for defining the *Ürün* class is to have a super-class that serves as a collection of all classes and individuals of the ontology, in other words, serves as a pool of classes and objects. Definition of *Ürün* class is necessary and also supports flexibility for querying by using short query sentences for any product that is having any individual, data property or relation; and it is also necessary for assigning data properties or relations for all or classes and their individuals at one time. For example, searching for any class having *Seviye* value “2” or a relation like *hasTip* or a keyword data as *ipek* will be an easy task by querying only the *Ürün* class. Class type definitions are done on class individuals which mean that defining an individual type also defines the class type. Due to the mentioned advantage of the *Ürün* class, each individual is defined in its own type and *Ürün* class type, by this way; classes are also defined in both their own class types and *Ürün* class type. Individual type definition which is also the class definition is pictured in the following Figure.

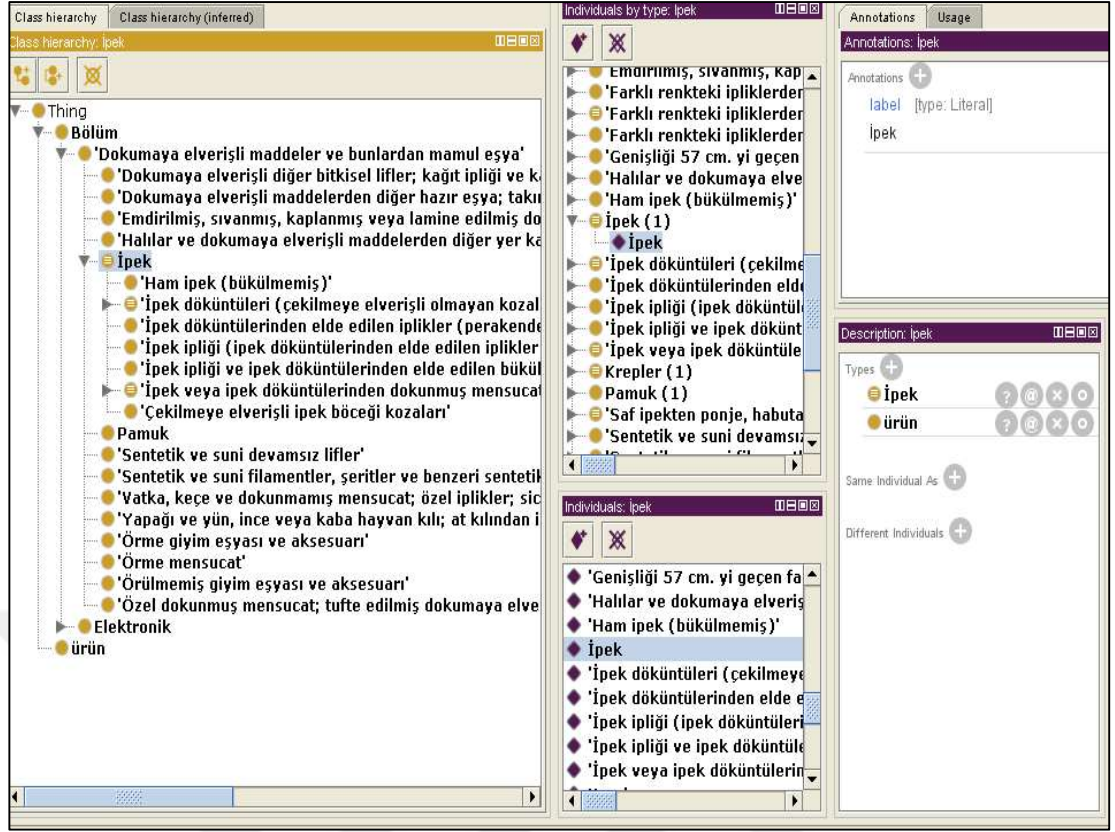


Figure 28. Individual/Class type definition.

Complexity of the ontology increases with the number of classes, individuals, properties and relations. This kind of class type definition, described above, also keeps number of classes, individuals, relations and properties at a minimum number which decreases complexity. Therefore, by using minimum number of entities, efficiency in traceability on ontology entities is gained and efficiency in querying system is also supported.

In addition, two types of relations and their inverses are decided to construct between the classes and the subclasses. *hasÜye* relation with its inverse *isÜyeof*, and *hasTip* relation with its inverse *isTipof* which are all transitive. The relations *hasÜye* and *hasTip* relate the individual to the individuals of its subclasses; and *isÜyeof* and *isTipof* relations relate the individuals to the individual of its superclasses.

The constructed ontology design for this study is figured by VoWL plugin of Protégé tool and displayed in Figure29, 30 and 31.

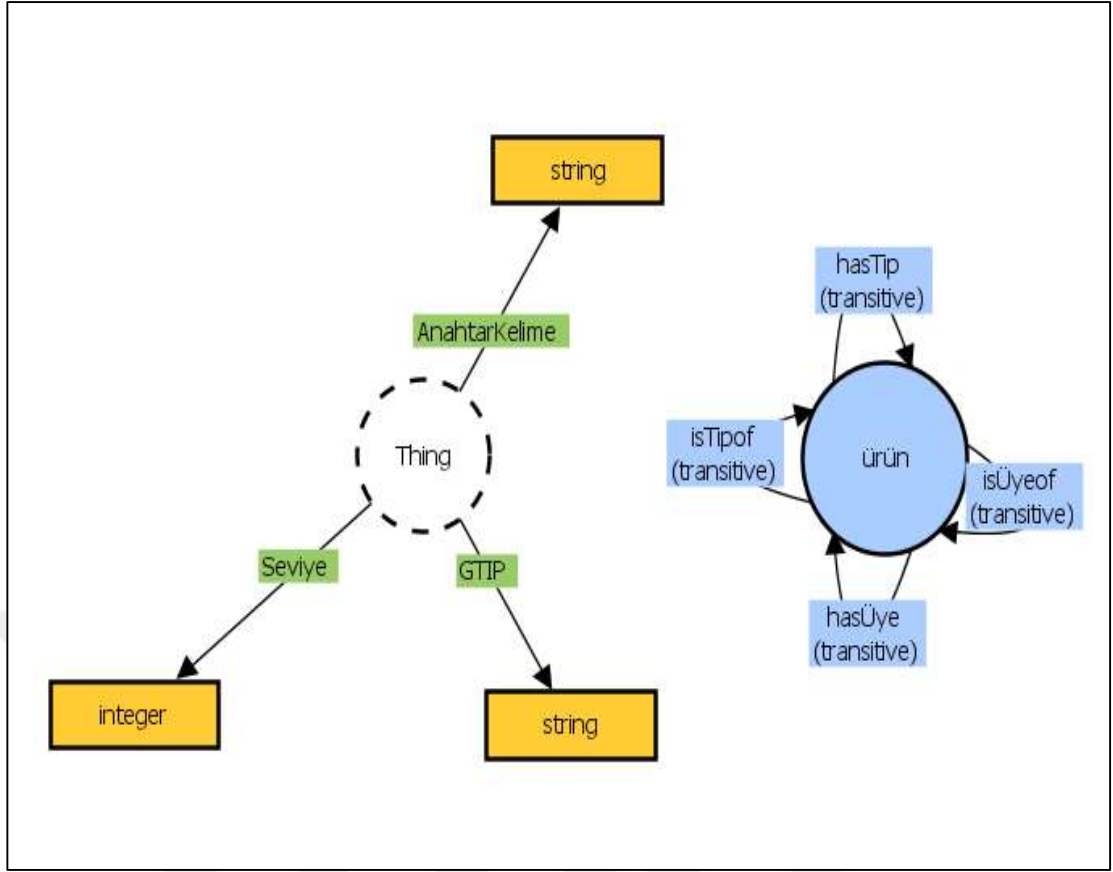


Figure 29. Ontology structure graph-1.

As depicted in Figure 2, the root class is Bölüm 11 namely *Dokumaya Elverişli Maddeler ve Bunlardan Mamul Eşya*. Chapters of *Dokumaya Elverişli Maddeler ve Bunlardan Mamul Eşya* are defined as subclasses like *İpek* class.

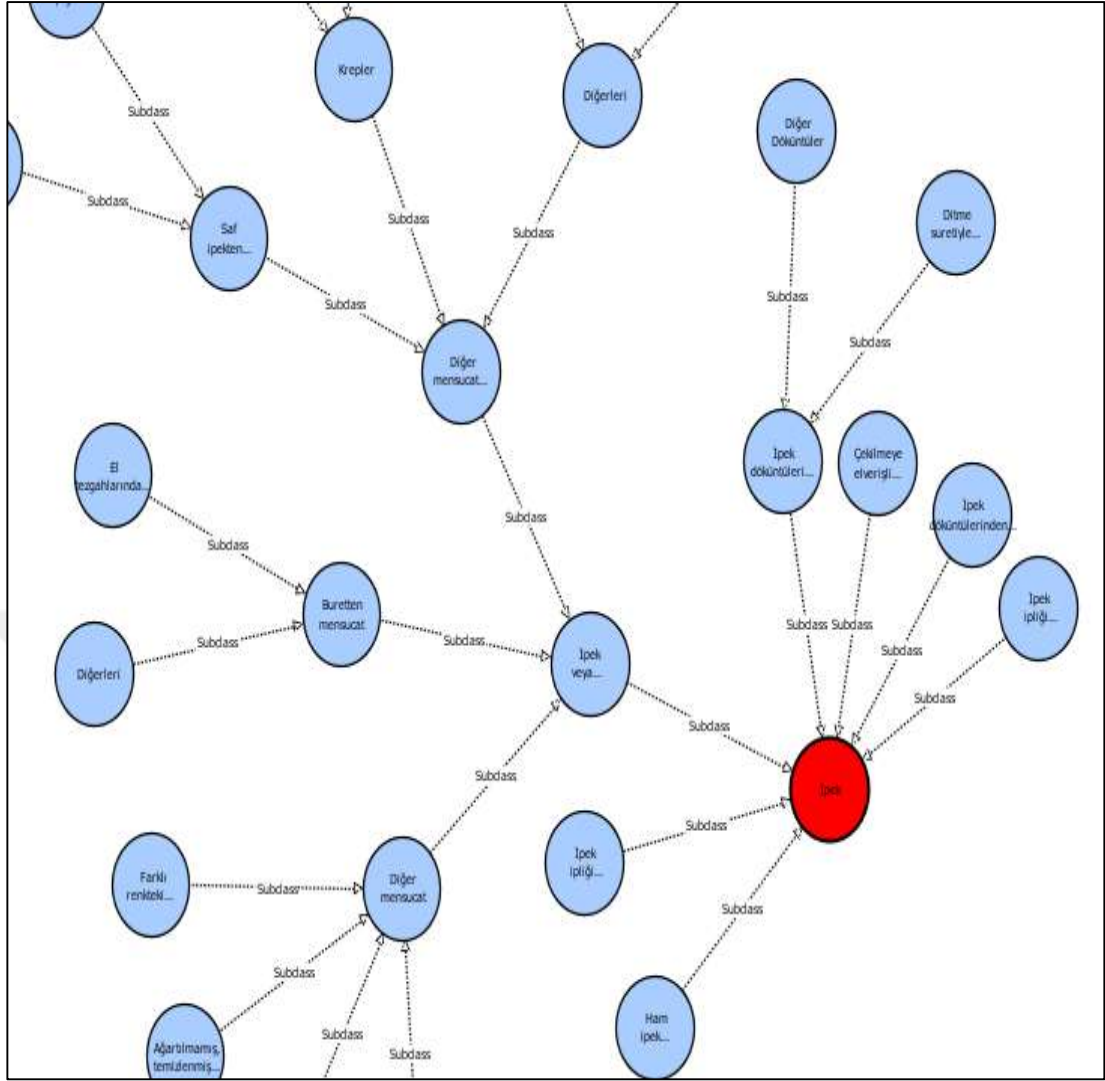


Figure 31. Ontology structure graph-3 / *İpek* Chapter hierarchy.

3.2 Studies on NLP

In this thesis, the Zemberek tool is used for the application of NLP techniques to extract keywords. Keyword injection to the ontology is another important issue where different methods and four different ways are applied to inject the keywords. The work on keyword extraction and keyword injection is detailed in the following sections:

3.2.1 Keyword Extraction using NLP

In order to satisfy the thesis objectives, understanding the morphological structure of Turkish NLP tools and particularly Zemberek, are the main steps of this thesis.

Constructing an ontology with the integration of an NLP tool for morphological analysis is the novelty of this thesis. Inevitably, this approach also supports the ontology querying where extracting information from the query supports structure providing a special solution for the challenges of querying. This perspective will hopefully provide a new understanding and viewpoint for the Turkish ontology designers.

Recall that the system (HS) currently consists of 5000 groups of products each described by six-digit codes, then an additional six-digits, refers to a particular product from the referred group, totaling to a twelve-digit HS code (G.T.İ.P.). In this study, the constructed ontology consists of classes and individuals which have their own keywords, namely *AnahtarKelime*, that are descriptive words of the related product. As an example the product *Çekilmeye elverişli ipek böceği kozaları* is defined with the keywords *çekilme*, *elveriş*, *ipek*, *böcek* and *koza*. Determining keywords using NLP techniques is also a necessary operation for improving the querying of the ontology and finding the correct G.T.İ.P. code for the users. In order to specify the keywords, The Turkish NLP tool called Zemberek is used for the morphological analysis of Turkish keywords. Keywords are extracted from the name of classes and description of the products represented by the classes.

Challenges in Keyword Specification

Because Turkish is an agglutinative language, Turkish words are made up of roots and affixes which need to be morphologically analyzed by extracting the different suffixes to reach the root. As stated before, morphological analysis of the words is done by using Zemberek. Some examples of the morphological analysis results are presented in Table 6:

Table 6. Example of morphologically analyzed keywords.

Word	Morphological Analysis
ağartılmamış	[[Kok: ağar, FIIL] Ekler: FIIL_OLDURGAN_T + FIIL_EDILGEN_IL + FIIL_OLUMSUZLUK_ME + FIIL_DONUSUM_MIS]
döküntüleri	[[Kok: döküntü, ISIM]Ekler: ISIM_COGUL_LER + ISIM_BELIRTME_I]

Table 6. Continued.

bükülmemiş	[[Kok: bük, FIIL] Ekler: FIIL_EDILGEN_IL + FIIL_OLUMSUZLUK_ME + FIIL_DONUSUM_MIS]
çekilmeye	[[Kok: çekil, FIIL] Ekler: FIIL_OLUMSUZLUK_ME + FIIL_ISTEK_E]
renkteki	[[Kok: renk, ISIM]Ekler: ISIM_KALMA_DE + ISIM_BULUNMA_KI]
Ponje	[[Kok: ponje, ISIM]]
temizlemeden	[[Kok: temizle, FIIL] Ekler: FIIL_OLUMSUZLUK_ME + FIIL_OLUMSUZLUK_DEN]
boyanmış	[[Kok: boya, FIIL] Ekler: FIIL_EDILGENSESLI_N + FIIL_DONUSUM_MIS]
dokunmuş	[[Kok: dokun, FIIL] Ekler: FIIL_GECMISZAMAN_MIS]
karıştırılmamış	[[Kok: karış, FIIL] Ekler: FIIL_ETTIRGEN_TIR + FIIL_EDILGEN_IL + FIIL_OLUMSUZLUK_ME + FIIL_DONUSUM_MIS]

The challenges of Turkish and the solutions presented are given below:

Negative suffix of words:

The suffix *-me/ma* can be seen in Turkish grammar sometimes as a negative suffix that gives negative meaning, or as a derivative suffixes that changes the root meaning to a passive form, or to change a verb to a noun. This causes an ambiguity in Turkish NLP. Some examples are listed in Table 7, in which the word column represents some examples of words used for the input, and Morphological Analysis column stands for the output of Zemberek. The third column, Zemberek Root column, shows the roots suggested by Zemberek, and the last column named Suggested Root includes the keyword constructed by human intervention

Table 7. Morphologically analyzed word examples.

Word	Morphological Analysis	Zemberek Root	Keyword
çekilmeye	[[Kok: çekil, FIIL] Ekler: FIIL_OLUMSUZLUK_ME + FIIL_ISTEK_E]	çekil (reeling + negativity)	çekilme (Passive) (reeling)

Table 7. (Continued).

Ditme	[[Kok: dit, FIIL] Ekler: FIIL_OLUMSUZLUK_ME]	dit (verb) (comb+negativity)	ditme (noun) combed
-------	---	---------------------------------	---------------------------

In order to resolve the ambiguity mentioned above, that if the *-me/ma* suffix gives the negative meaning or changes the meaning of root to a passive form, given in Table 6, the suffix *-me/ma* is considered as a part of the root, and the root is injected in keywords in the following form:

[[Dit]+[me]]=[ditme]. Additionally, this ambiguity resolution is taken into account when the possible user queries are being parsed by the developed system.

Passive root meanings:

Since, each keyword is directly produced from the entity names of the ontology, no matter what the structure of the root is, such as root+me/ma (negativity), root+me/ma (passive form), it is related to an entity of the ontology. Therefore meaning of *-me/ma*, as a suffix, is ignored and considered as part of the root as shown in Table 8 where the word column represents some example words of the input, and Morphological Analysis column stands for the output of Zemberek. The third column, Zemberek Root column, shows the roots suggested by Zemberek, and the last column named Suggested Root includes the keyword constructed. Note that, for any word which does not include any ambiguous suffixes, Zemberek's suggestion is taken into account.

Table 8. Morphologically analyzed word examples.

Word	Morphological Analysis	Zemberek Root	Keyword
ağartılmamış	[[Kok: ağar, FIIL] Ekler: FIIL_OLDURGAN_T + FIIL_EDILGEN_IL + FIIL_OLUMSUZLUK_ME + FIIL_DONUSUM_MIS]	ağar: (going white)	ağartılma (Passive) (being bleached)

Table 8 (continued).

bükülmemiş	[[Kok: bük, FIIL] Ekler: FIIL_EDILGEN_IL + FIIL_OLUMSUZLUK_ME + FIIL_DONUSUM_MIS]	bük (twist)	bükülme (Passive) (being twisted)
------------	--	----------------	---

3.2.2 Keyword Injection

As stated before, Protégé defines relations (object properties) and data properties only for individuals, not for classes. As classes do not have any relations with each other directly, they relate by their individuals using object properties. On the other hand, the other property definition, namely, data property is used to relate individuals to literal data (strings, numbers, integers, etc.). Two more definitions exist for relations and literal data that are Equivalent class definition and SubClassOf restriction which are also restrictions in class definition.

In ontology construction, locating any entity in the ontology or specification of any entity can take place in a number of different ways. For example, locating an entity under a specific class or giving a value to this entity will differentiate the individual from the others.

In the study, a lexicon of 136 keywords which is presented in Appendix G is injected into the ontology. The keywords are used in a number of different ways; using keywords as class restrictions; using keywords as identifier for subclasses and superclasses of the target entity. Furthermore, the keywords are injected as data property. Usage of these keywords is the most critical part of the study. Therefore, a number of ontology designs are analyzed and constructed and also four different approaches are used for keyword injection. These approaches are detailed in Chapter 4.

CHAPTER 4

IMPLEMENTATION OF THE SYSTEM

4.1 Implementation

In this thesis, two different applications which are working separately are developed for the HS coding system. The first application focuses on keyword creation using Turkish NLP, and the second is developed for integration of three modules, namely, NLP libraries, ontology and user interface.

The first application is developed for creation of the keywords. Since the domain data has its own specific hierarchy due to the HS coding system, product specific keyword creation is done by analyzing the product names and description morphologically using Zemberek. The outcome of the application is related roots and affixes of the products names. Later, the roots are classified and the keywords are defined and injected to the ontology manually based on the hierarchical structure of the product groups.

The second application is developed for querying the constructed ontology by user's natural language. User inputs the description of the product in Turkish from the *GUI*; *Parser* function takes the input words, parses the keywords into the *QueryBuilder* function. *QueryBuilder* function analyses the keywords morphologically, processes some rules on the input words, creates a query sentence and sends it to *QueryEngine*

function. *QueryEngine* function uses HermiT reasoner for loading and querying the ontology with the inserted keywords. After reasoner gets the query result, *QueryEngine* function sends the result to the *Printer* function. *Printer* function analyses the result, checks for the special keywords in the result, removes unwanted products. At the end, results are served to the user by *GUI*. Each function of the application is explained in detail in the following sections:

System design is pictured in the following Figure and detailed information about the modules is presented in the following sections:

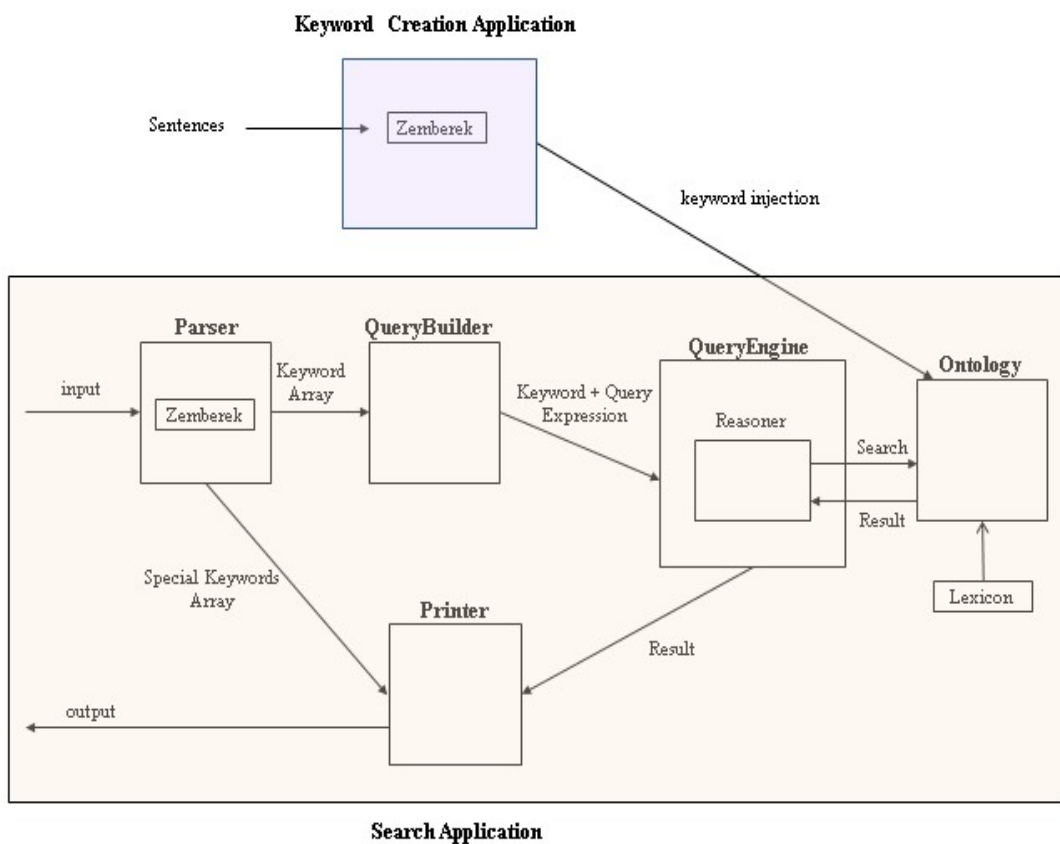


Figure 32. Components of the system.

4.1.1 Keyword Creation Using Turkish NLP

In this study, a Java application is developed to use Zemberek libraries to perform morphological analysis. Zemberek libraries are used for decomposing sentences into words and analysing words morphologically, where the outcome of this process is the root and suffixes of the words.

All class names and related information as explanation of products are listed in a text file in separate lines. The application reads each line from the input text file and outputs morphological analysis of each word into an output text file. Analyzing the output text file of the morphologically analyzed words and specifying the keywords from the analyzed words was challenging. Only using roots of the word as keywords is not a solution since suffixes change the meaning of the roots, so human intervention becomes a necessity about making decision of the keywords.

The following Figure depicts the data flows through the application:

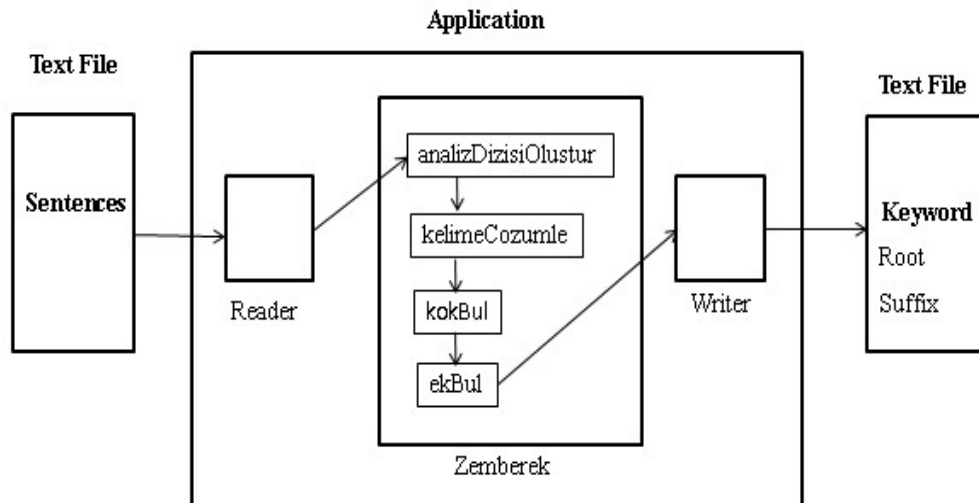


Figure 33. Data flow diagram of Keyword Creation application.

Firstly, a text file is created including all class names and related information such as the description of products, which are listed in Tables 3, 4 and 5 in Chapter3, in

separate lines. And also an output text file is created for the morphological analysis results. The application is developed in mainly six functions explained below:

- The *Reader* reads each line from the input text file and sends it to the function *analizDizisiOluştur* for morphological analysis of the sentence.
- *analizDizisiOluştur* function reads each line coming from the reader and splits the input sentence into its words and puts each word into a list for analysis using the Zemberek libraries.
- *kelimeÇözümle* function takes each word from the list and checks for the conjunctions and pronouns, then removes them from the list of words, since the conjunctions and pronouns are common words and are not specific keywords to any product. After words, *kelimeÇözümle* function fires *kokBul* and *ekBul* functions.
- *kokBul* function inputs each word, finds root of word and put them into an array of roots.
- *ekBul* function finds affixes of each word and puts these affixes into an array. *ekBul* function also analyzes types of the words and give special definitions to special affixes. As mentioned before, for the keyword creation, using only root of the word is not sufficient; affixes and the special descriptions are also necessary for making correct decisions about the keywords. An example about the affixes and their special definitions is provided below:

Word: "olmayan"
Affix: "-me/ma" affix
Affix output: "FIIL_OLUMSUZLUK_ME"
Special description: NEGATIVE

Word: "ipekten"
Affix: "-den" affix
Affix output: "ISIM_ÇIKMA_DEN"
Special description: MADE OF

Word: "ipekli"
Affix: "-li" affix
Affix output "ISIM_BULUNMA_LI"
Special description: HAS

Word: "suretiyle"
Affix: "-le" affix
Affix output "ISIM_BIRLIKTELIK_LE"
Special description: MADE BY

Word: "renkteki"
Affix: "-ki" affix
Affix output "ISIM_BULUNMA_KI"
Special description: HAS

Figure 34. Affixes and their special definitions in the application.

Writer function outputs each word, its root and affixes into another text file in different lines. An example of application outcome which is the morphological analysis of the words is presented in Figure 35 below. The first line indicates line number and the sentence in the input text file, the second line indicates root and affixes of the first word, the third line indicates root and type of root, fourth line writes the type of root only, fifth line indicates the special description of the affixes and the other lines indicate the other affixes.

```

15: Çekilmeye elverişli ipek böceği kozaları
. [ Kok: çekil, FIIL ] Ekler: FIIL_OLUMSUZLUK_ME + FIIL_ISTEK_E .
.. çekil FIIL ..
... FIIL_KOK ...
... FIIL_OLUMSUZLUK_ME ... -> NEGATIVE
... FIIL_ISTEK_E ...

. [ Kok: elveriş, ISIM ] Ekler: ISIM_BULUNMA_LI .
.. elveriş ISIM ..
... ISIM_KOK ...
... ISIM_BULUNMA_LI ... ->HAS

. [ Kok: ipek, ISIM ] .
.. ipek ISIM YUM ..
... ISIM_KOK ...

. [ Kok: böcek, ISIM ] Ekler: ISIM_TAMLAMA_I .
.. böcek ISIM YUM ..
... ISIM_KOK ...
... ISIM_TAMLAMA_I ...

```

Figure 35. An example of morphological analysis of the words.

At the end of morphological process, the next step is the investigation of the output file by analyzing root of the words to see whether they represent the correct meaning for the keyword. As mentioned in the previous Chapter, investigating the output text file for the morphologically analyzed words, and specifying the keywords was challenging. Only using roots of the words as keywords is not a solution since suffixes change meaning of the roots; so human intervention becomes a necessity about making decision of the keywords. Roots and the suffixes are investigated together to form keywords to overcome the challenges as negativity and *-me/ma* suffixes that change the meaning of the roots. Finally, keywords that represent the products are created as mentioned in Chapter 3.

4.1.1.1 *Keyword Injection*

Keywords are injected to the ontology manually after the creation of keywords. Keyword injection is the most important part of this study. As the HS coding system has its own specific hierarchical structure, all classes, subclasses were individuals are kept as they are, but defining relations between these entities, and inheriting the keywords to the subclasses and individuals is also challenging. To inject the keywords; entities, relations and data properties are specified firstly, secondly relations between the entities are constructed and finally the keywords are injected. Detailed information about keyword injection process is explained in the following sections:

4.1.1.2 *Specifying Entities, Relations and Data Properties*

Ontology construction is done using Protégé 4.3 which defines relations as object properties and data properties are defined only for objects, called individual, and not for classes. So, after long hours of study and many trials of ontology construction, it is decided that, each class must have its own individual for defining the relations and data properties of the class. If individuals of the same class are different and not related to each other, this is also defined in ontology.

Firstly, each individual is defined with the same name of its own class; this avoids confusion in connecting an individual to its own class as a member and also it is easy to trace entities in the ontology. A label containing the name of the individual is also inserted as annotation to the individual, and this is also done for classes. Secondly, each individual is defined both in type of its own class and Ürün class for keeping the number of individuals of ontology at a minimum as mentioned in the previous section. Finally, relevant data of classes are defined for the individuals as data properties which are used to define and relate individuals to literal data (e.g., strings, numbers, integers, etc.). There are three data properties defined for each individual, namely *AnahtarKelime* is defined for keywords for describing the product, *GTIP* for HS code and *Seviye* for level of product in the hierarchy. *AnahtarKelime* and *GTIP* data properties are defined as string type and *Seviye* data property is defined as integer type. Defined data properties are shown in Figure 36.

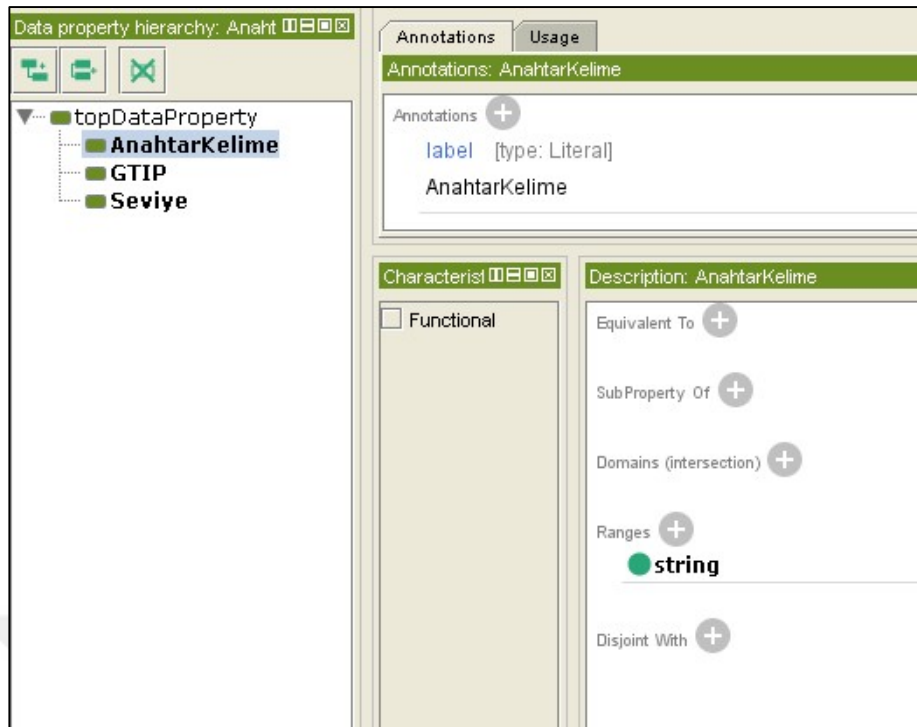


Figure 36. Data property definitions of individuals.

Data properties are defined on individuals, besides they are also defined by SubClassOf restriction, and are explained in the following sections.

4.1.1.3 Constructing Relations

The ontology tool Protégé has two types of property definition: Object property and Data property. As mentioned in the previous chapter, classes do not have relations with each other directly, they relate to each other with their individuals. To relate an individual to the other, object properties are used. As shown in Figure 3, four relations are defined for each individual. The relations *hasTip* (hasType) and *isTipof* (isTypeof) are inverses of each other and both are transitive such that subclasses also have the same relation. Domain and range values are also defined. The other relations *hasÜye* (hasMember) and *isÜyeof* (isMemberof) relation, are inverses of each other and are both transitive. The relations *hasÜye* and *hasTip* relate the individual to the individuals of its subclasses; and *isÜyeof* and *isTipof* relations relate the individuals to the individual of its superclass. These relations are shown in the following Figure:

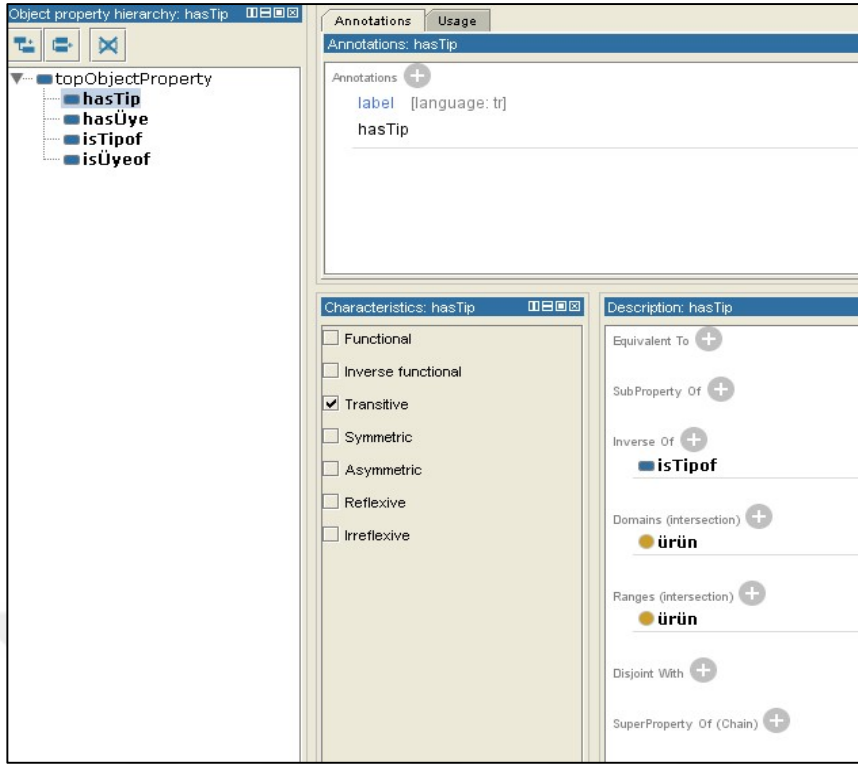


Figure 37. Object property definitions of individuals.

In the following Figure, description, label, types, object and data properties of the individual named *Burretten mensucat* is presented:

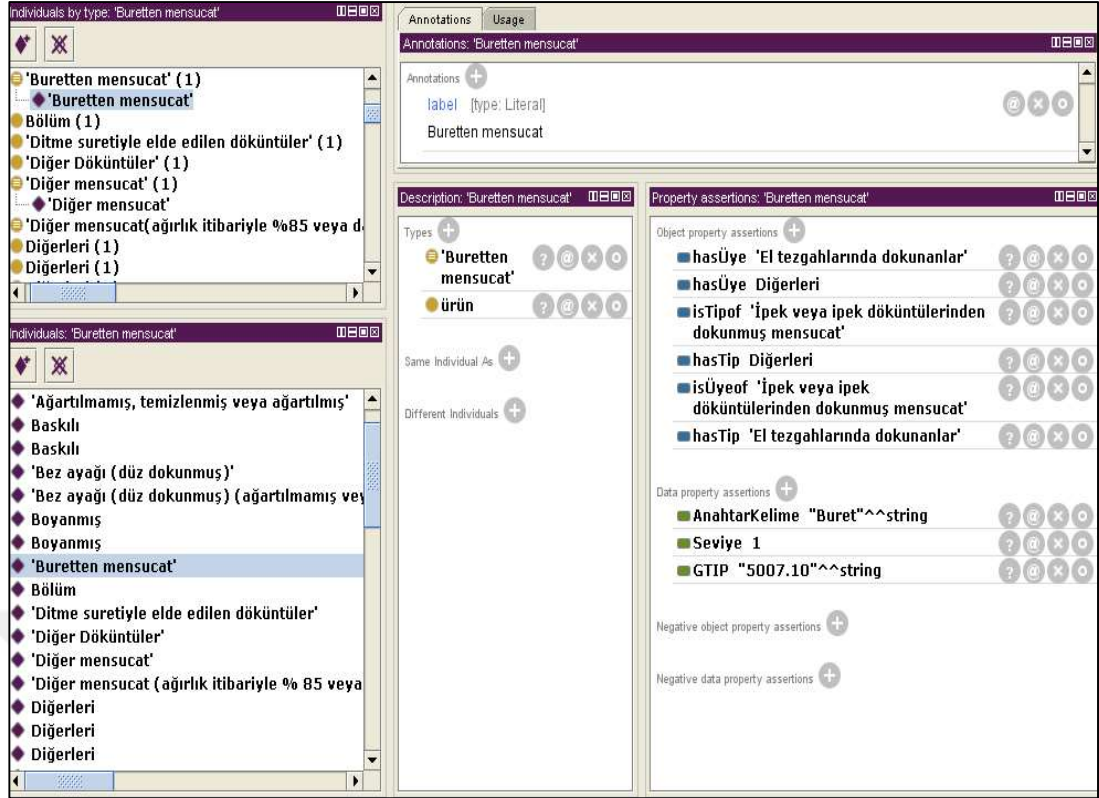


Figure 38. An individual definition and its data properties.

4.1.1.4 Keyword Injection as Data Property

Keywords of each individual are defined as *AnahtarKelime* data property. *AnahtarKelime* value is defined as *string* type as shown in Figure 36. After the injection of keywords, the number of entities and relations of the ontology are not changed; only the number of literal information for each individual is increased. Complexity and scale of ontology also stands the same. Query response time is the best of all four types of constructed ontologies and query results were correct. As *AnahtarKelime* values are specific to its individual, the information inheriting from the superclasses cannot be reached directly. On the other hand, there is a limitation in this type of design such that, each keyword belonging to a superclass should be defined over again for its subclasses. This means, number of a subject keyword definition in superclass is also multiplied by the number of subclasses since that it is also defined in the subclasses. As an example represented in Figure 39, although the keyword *ipek* is defined as *AnahtarKelime* data property in *İpek* class, it is defined in its subclasses additionally. To overcome this restriction, besides using data properties

for keyword injection, SubClassOf restrictions are also used and will be explained in the following sections.

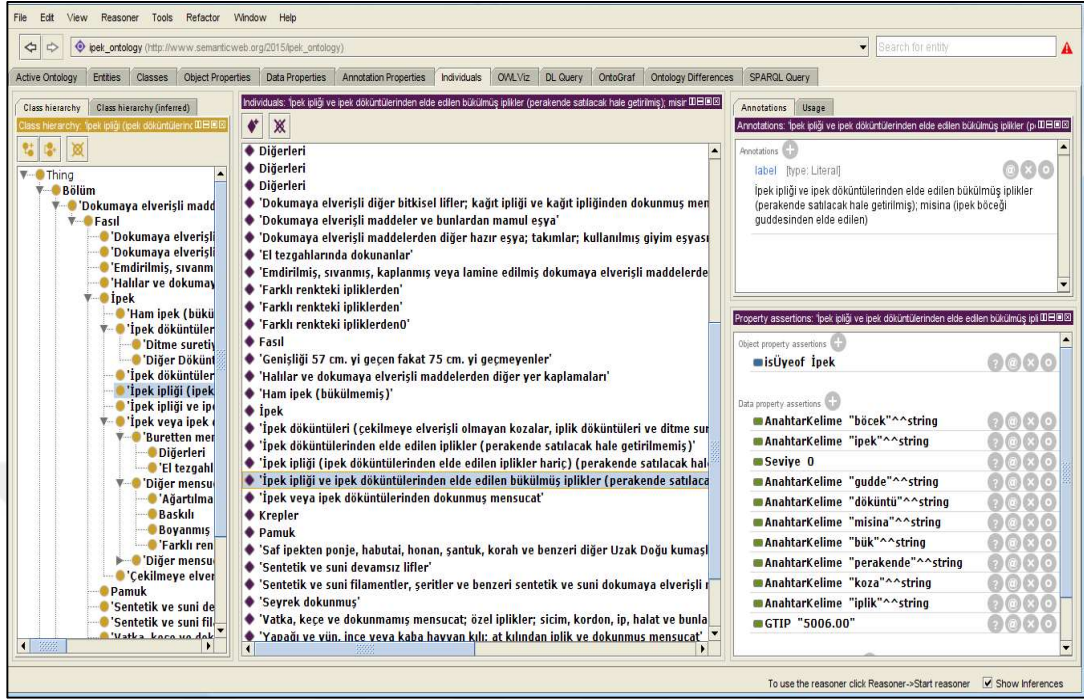


Figure 39. Keywords defined as data property.

4.1.1.5 Keyword Injection as Object Property

To inject the keywords using object property definition, an *Anahtar* class is defined and all the specified keywords are defined as individuals, as members of *Anahtar* class which is shown in Figure 40. The Object properties *hasAnahtar* and *isAnahtarof* are defined for relating each individual to the keyword individuals. Defining keywords as individuals increases the number of entities and number of relations in the ontology according to the number of keywords so complexity of the ontology is increased. The newly constructed ontology is queried by the developed system, and as the query results were correct, the query response time increased.

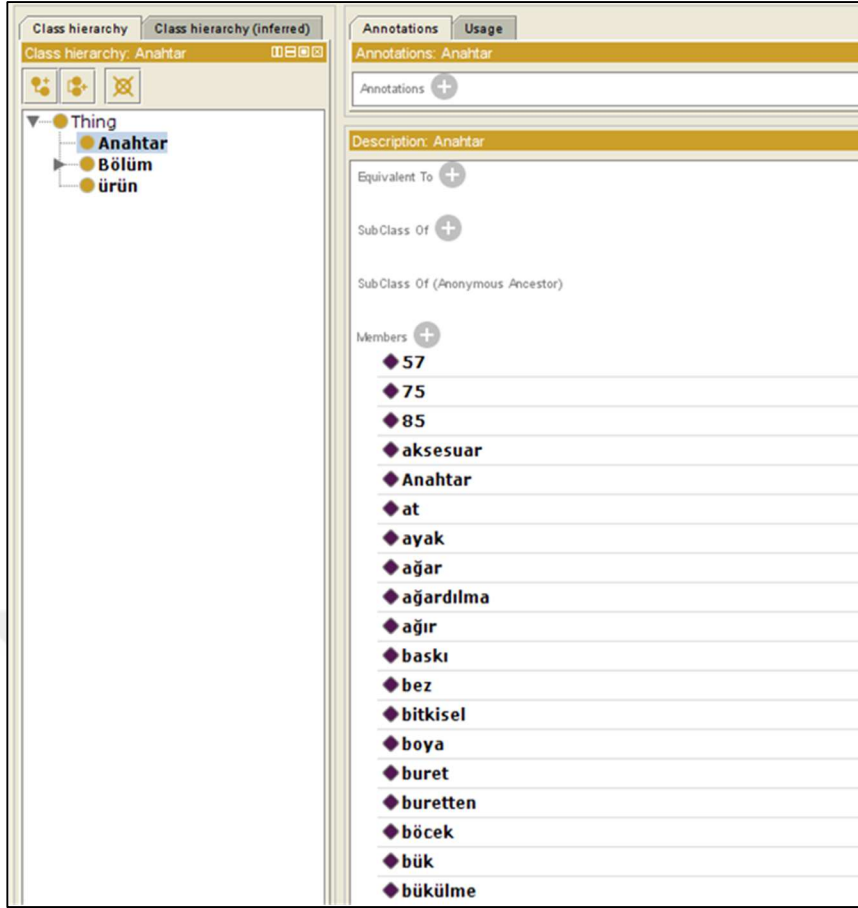


Figure 40. Anahtar class and its members.

As a result, according to the query response time, injecting keywords as object property is not efficient for our ontology structure and this design is disregarded.

4.1.1.6 Keyword Injection as Equivalent Class Definition

Equivalent class definitions are for defining the classes having the same individuals and constructing the relations as restrictions. They are implied to inherit the subclass relationship in both directions between superclass and the subclasses. This means, *AnahtarKelime* definition of a superclass inherits to the subclasses as though they have the same defined *AnahtarKelime* value, reciprocally, *AnahtarKelime* defined in the subclass as Equivalent class description is also *AnahtarKelime* value of the superclasses. An individual has many different *AnahtarKelime* values, besides, an *AnahtarKelime* value can be defined in different individuals that mean, *AnahtarKelime* values are not specific to individuals, and they are common for use.

Equivalent class description is efficient in ontology construction, but is not appropriate for our ontology. Because this approach presumes all individuals as the same individual, if they have the same *AnahtarKelime* value. So all individuals are supposed as the same, as if there is only one individual, this means relating each individual to each other and query on all individuals who is not efficient.

An example of Equivalent class definition is represented in Figure 41. As pictured in the Figure, *Burettten mensucat* class has equivalent class definition with *AnahtarKelime* value *burettten*, and *AnahtarKelime* values related to Equivalent class definition of *BuretttenMensucat* class are inferring from its superclasses which are displayed in SubClassOf (Anonymous Ancestor) part in the Figure. *AnahtarKelime* values infers to the subclasses of *Burettten mensucat* with the *AnahtarKelime* value *burettten* inserted to this class as equivalent class description.

Equivalent class definitions support querying individuals, equal classes, super and subclasses of a class directly without doing computation, keeps the number of entities the same but increases complexity and reasoning time. As a result, this design is also discarded.

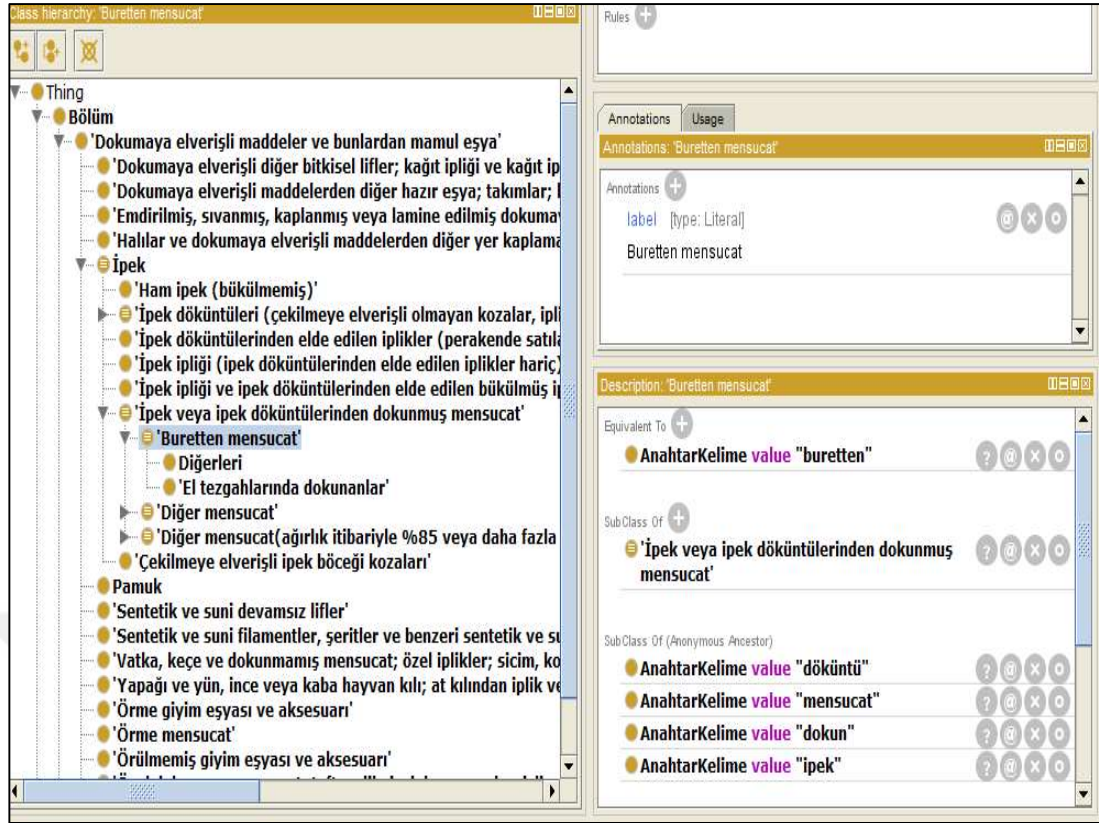


Figure 41. An example of Equivalent class definition.

4.1.1.7 Keyword Injection as SubClassOf Restriction

SubClassOf restriction is used for defining the classes as subclass of defined class as expression and constructing the relations as restrictions. SubClassOf restrictions are implied to infer the subclass relationship in one-way direction from superclass to its subclasses. This means, *AnahtarKelime* definition of a superclass inherits to the subclasses as though they have the same defined *AnahtarKelime* value, but *AnahtarKelime* defined in the subclass is not *AnahtarKelime* value of the superclasses. Contrary to Equivalent class description, SubClassOf restriction does not relate all individuals to each other as the same individual; it only relates all individuals of the superclass to the individuals of the subclasses.

An example of SubClassOf restriction is represented in Figure 42. As pictured in the Figure, *Buretten mensucat* class having SubClassOf restriction with *AnahtarKelime* value *buretten* and *buret*; and *AnahtarKelime* values related to SubClassOf restriction of *Buretten mensucat* class are inherited from its superclasses which are

displayed in Subclass of (Anonymous Ancestor) part in the Figure. All the SubClassOf restrictions on *AnahtarKelime* value, both from the super class and from *Burettten mensucat* class, also infers to the subclasses of *Burettten mensucat*.

SubClassOf restriction supports reasoning individuals, superclasses and subclasses of a class directly without doing computation, keeps the number of entities the same and has fewer but beneficial relations. Complexity and reasoning time are satisfying so this approach is chosen for the system. As inferring keywords from superclasses to subclasses by SubClassOf restriction, some more keywords are also defined to the classes that do not have any subclass. To overcome this difficulty, keywords are defined as Data property to these classes.

Finally, in this study, both SubClassOf restriction and Data property definitions are used for injecting keywords as *AnahtarKelime* values.

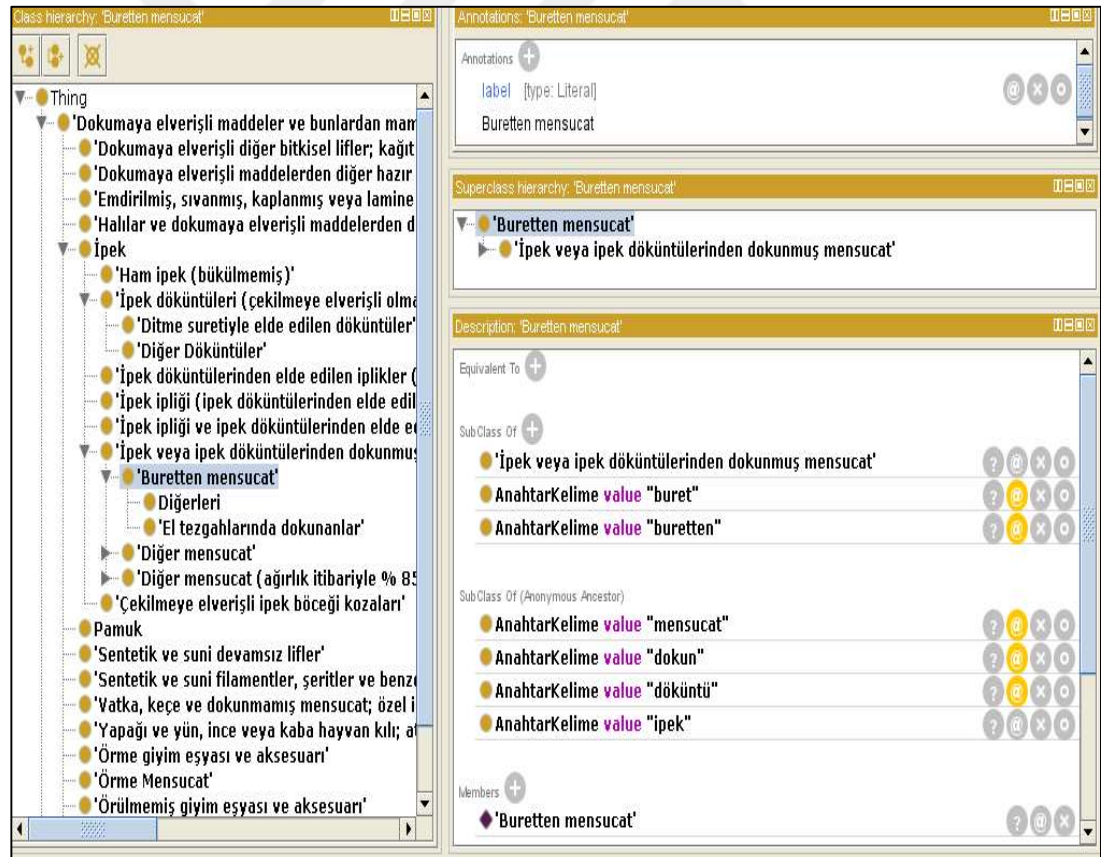


Figure 42. As an example of SubClassof restriction.

In the following section, the second application which is the main outcome of the thesis is explained:

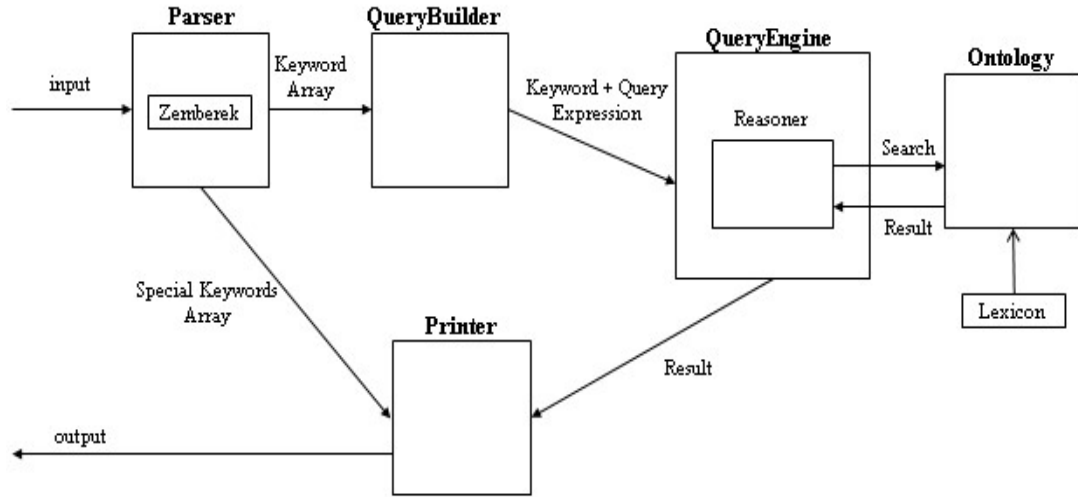
4.1.2 G.T.İ.P. Search Application

As a part of the thesis, *G.T.İ.P. Search Application* is developed in Java in Eclipse Mars.2 Release (4.5.2) on Windows 8. Zemberek is used for parsing the input sentences and the words, HerMiT 1.3.8. is used as reasoner and OWL API is used for loading and querying the ontology.

This application is developed in mainly six parts. The following sections are going to present detailed information about the implementation; however, giving an overview will provide better understanding for the novel structure. The part of the system is given as follows:

- The user interface where the user queries are accepted and results are presented.
- The *Parser* where user input is reduces to its roots and checked to see whether the given sentence has a particular negative word such as “olmayan”, “hariç” and etc. Additionally, parsed words are checked for any negativity suffixes such as “siz”, “me/ma” and etc.
- *Query Builder* which uses parsed words and creates a query sentence for the ontology.
- *Query Engine* where the incoming query sentence is used for reasoning the ontology.
- Ontology module.
- *Printer* where all the result are checked for special words, the results are ranked and sent to the user interface.

The following Figure depicts the data flows through the application.



Search Application

Figure 43. Data Flow Diagram of Application.

The main goal of the developed application is querying the constructed ontology with user's natural language which describes the products appearance. As pictured in Figure 43 above, by using a *GUI*, user inputs the description of the product in Turkish; *Parser* function takes the input words, parses the keywords into the *QueryBuilder* function. *QueryBuilder* function analyses the keywords morphologically, processes some rules on the input words, creates a query sentence and send it to *QueryEngine* function. *QueryEngine* function uses HerMiT reasoner for loading and querying the ontology with the inserted keywords. After reasoner gets the query result, *QueryEngine* function sends it to *Printer* function. *Printer* function analyses the result, checks for the special keywords in the result, removes unwanted products. At the end, results are served to the user by *GUI*. Each component of the application is explained in detail in the following sections:

4.1.2.1 *Parser*

In Figure 6 above, the search sentence of the user as input data is processed in the *Parser* function. Using Zemberek libraries, *Parser* inputs each line into a list for analysis by *analizDizisiOluştur* function. *kelimeÇözümle* function splits sentence into words. *kokBul* functions inputs each word, finds root of word and *ekBul* function

finds suffixes of the word. The outputs of this process are, root and affixes of the input sentence. Mainly, two different arrays are constructed by the *Parser* and the following processes take place:

- Affix checking:

Affix *-me/ma* is checked for ambiguity such that if the affix *-me/ma* exists then it is added to root and root is replaced to [root + me/ma] in input array.

Affix *-siz* is checked for negativity of word meaning. If *-siz* affix exists, the root of the word is stored in special keywords array where it is used in the process of *Printer* function. The words having *-siz* affix are eliminated in *Print* function so they are not displayed in the result.

- Extracting Negativity Information: If the root array includes specific words such as *hariç* and *olma* then the previous word's root is added to the special keywords array for not being displayed in the *Print* function.

After the processes take place, the input array is sent to *QueryBuilder* function for querying. Special keywords array is sent to the *Print* function for not displaying the entities having keywords listed in the special keyword array. At this point, giving additional information about the special issue of *NOT* querying will be informative and can indicate one of the key points of this thesis.

Firstly, there is no chance to make a query to the ontology for an entity that is not represented in ontology, meaning, non-existing entities cannot be queried. More technically, querying using *NOT* expression is not possible. As a result, the algorithm is developed to exclude entities from the result. Note that information of *NOT* comes from the morphological analysis of the sentence given by the user. So with the help of morphological analysis and an additional algorithm, *NOT* problem is resolved.

Data flow fragment of the *Parser* can be seen in Figure 44.

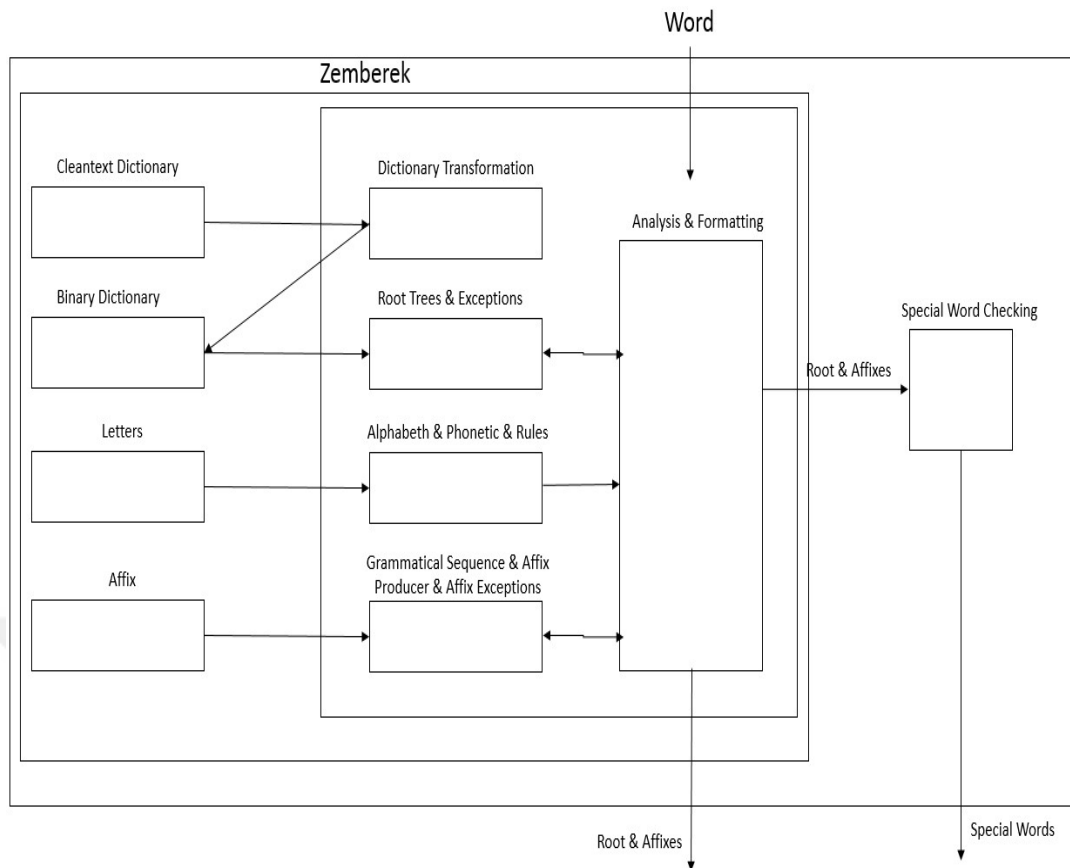


Figure 44. Data Flow Fragment of *Parser*.

4.1.2.2 *QueryBuilder*

QueryBuilder takes inputs as the input array and constructs a proper DL query sentence by appending specific query expressions; and sends the DL query sentence to the *QueryEngine*. Data flow fragment of *QueryBuilder* can be seen in Figure 45. In Figure 46, the procedure of the query sentence creation with input keywords is given as pseudo code:

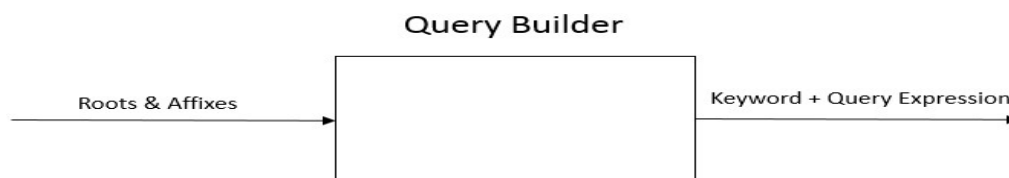


Figure 45. Data Flow Fragment of *QueryBuilder*.

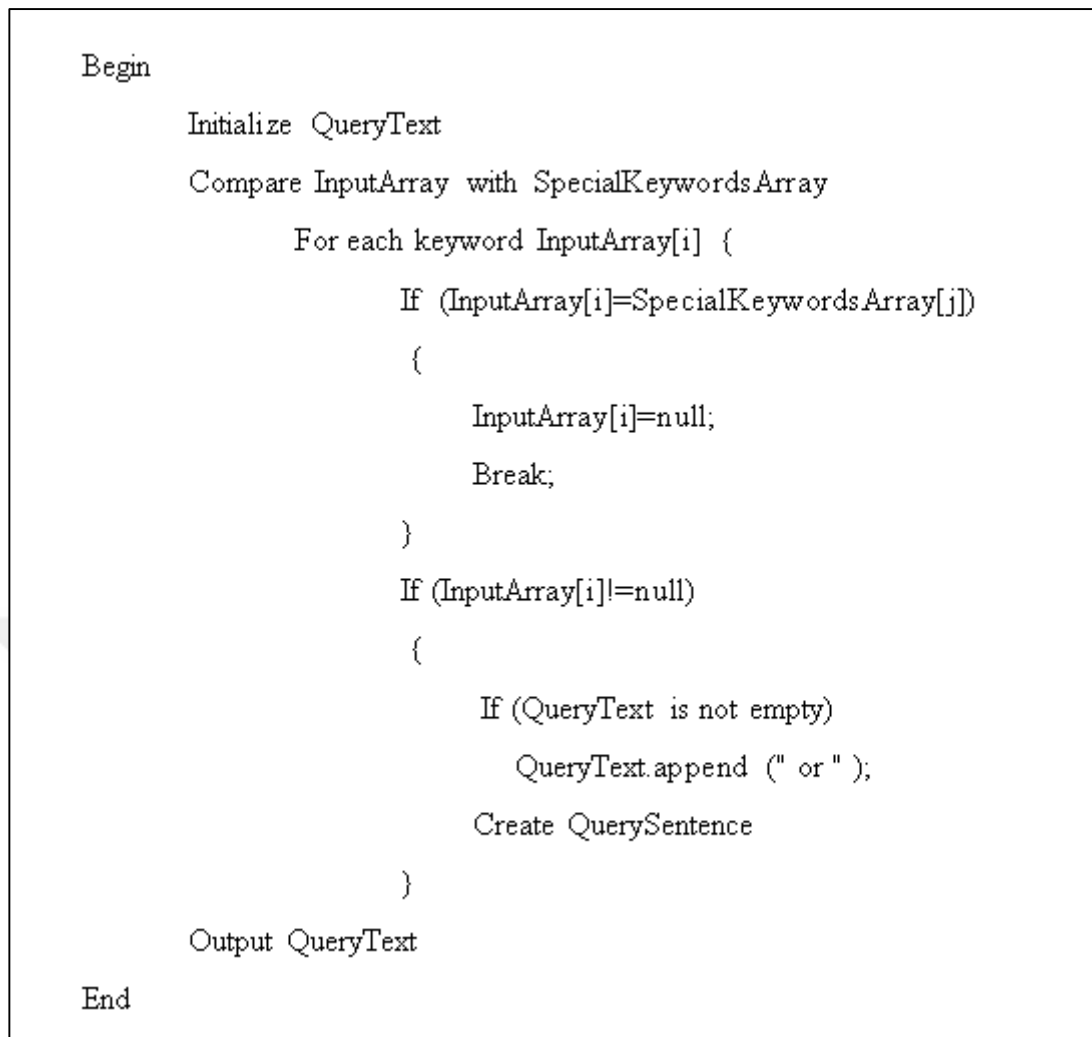


Figure 46. Pseudo Code of *QueryBuilder*.

As stated before, ontologies cannot be queried for non-existing entities, meaning *NOT* expression cannot be used in query sentences. That is why in the developed application, all the keywords in the input array are queried using *OR* operator and the classes having the keywords in the special keywords array are filtered out by *Printer* function. This is the only way to eliminate the entities that are not to be taken as the result.

4.1.2.3 *QueryEngine*

QueryEngine firstly connects to ontology and loads the ontology from file, and then initializes Hermit reasoner, precomputes interferences and checks for inconsistency in the ontology. As the ontology is consistent, Hermit reasoner processes the query

sentence and searches for the entities in the ontology. Ontology responds to the query with a set of entities/individuals which have the keywords corresponding to the input query. *QueryEngine* sends the matching entities to *Printer* function for printing and filtering the exceptional entities. Data flow fragment of query engine can be seen in Figure 47.

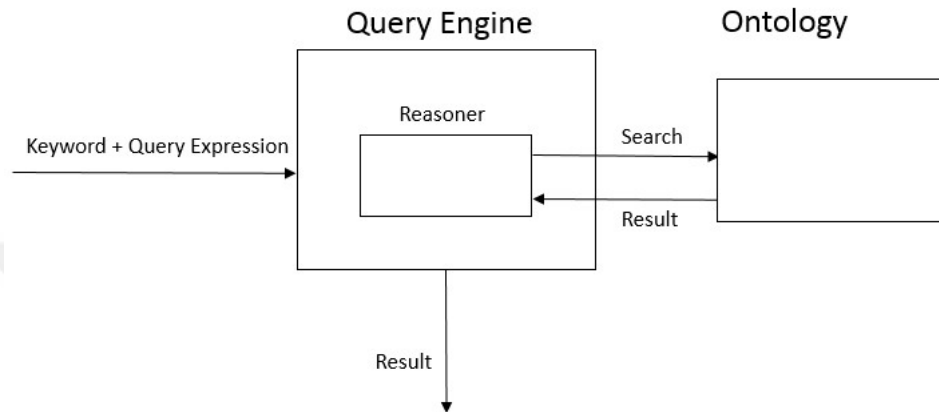


Figure 47. Data Flow Fragment of *QueryEngine*.

4.1.2.4 *Printer*

Printer gets the query result from *QueryEngine* and also special keywords array is sent to *Printer* by the *Parser*. Main function of *Printer* is taking the query result, comparing and filtering out the result with the special keywords array and sending the result to the *GUI*. Additionally, *Printer* computes matching keyword's counts as *Rank* to display in descending order with respect to *Rank*. Furthermore, this function calculates the processing time, total number of matching entities, subclasses and superclasses of entities and displays this information in the user interface, the reason for displaying superclasses and subclasses of entities is that; some of the product classes do not have *G.T.İ.P.* number and also some of the products have the same name such as *Diğerleri*; so user can define the *G.T.İ.P.* number for the product that they are searching for, by checking superclass *G.T.İ.P.* numbers or the subclass *G.T.İ.P.* numbers. Data flow fragment of *Printer* can be seen in Figure 48.

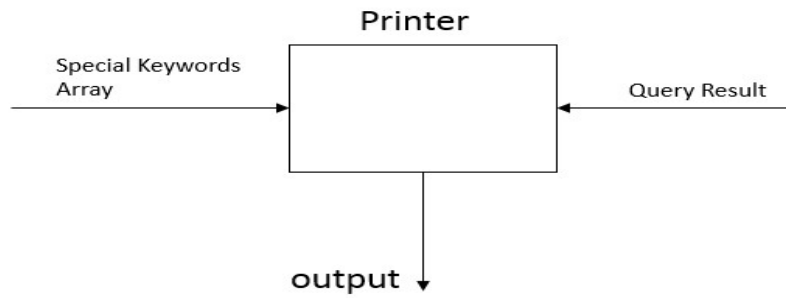


Figure 48. Data Flow Fragment of *Printer*.

As mentioned in the previous chapters, ontology hierarchy and keyword inheritance is provided by using SubClassOf restriction. SubClassOf restriction is also in simple text format as restriction; as an example [*AnahtarKelime value "ipek"*] class expression is added to the relevant class as restriction so that the keyword value *ipek* is inferred to the subclasses. Also some keywords are inserted as data property into the classes which do not have subclasses and have special keyword values unique to them.

In this study, using SubClassOf approach provides advantages which decrease the number of inserted keywords and relations, and also scale of the ontology. Although using this kind of approach is effective and easy to trace the keyword specifications and their inheritance in the ontology, it is not that easy to retrieve keyword information from class definition and filter them out. This kind of approach is newly used, and there exists no similar ontologies represented in this way. With this approach, the study diverges from the other studies on ontologies especially in research on Turkish ontology creation with NLP. This is the main point and the contribution of the thesis.

As mentioned before, each class has its own individual and keywords inserted to this individual. Using SubClassof restriction, keywords inherit to the subclasses and their individuals. Eventually, retrieving keyword value from the SubClassOf restrictions is a challenging issue. The *Printer* function overcomes this challenge with the following steps:

- *Printer* gets query result which is collection of individuals from the *QueryEngine*.
- For each individual, retrieves all superclasses of these individuals.
- Checks for the inherited keywords and special keywords of the superclasses.
- Checks for keywords of the individual, if the keywords do not infer from the superclasses.

Prints the matching individuals and keywords and prints *Rank*.

The sequence diagram is figured in Figure 49 and pseudo code of *Printer* is pictured in Figure 50.

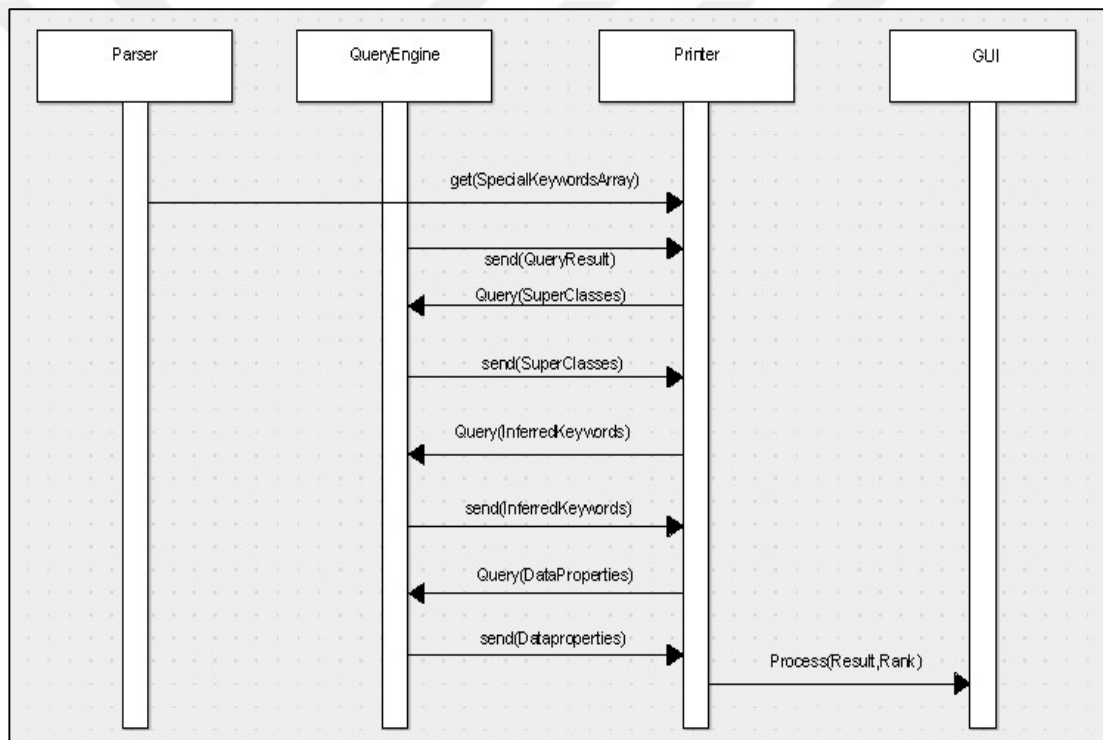


Figure 49. Sequence diagram of *Printer*.

```

Begin
Loop individuals
{
  Loop superclasses of individuals
  {
    Get direct individual of each superclass
    if (individual keywords==SpecialKeywordsArray[i])
    {
      break;
    }
    if (individual keywords==InputArray[i])
      Add class and keyword to Result
  }
  Get keywords as dataproperty
  if (keyword=Inputarray)
    Add class and keyword to Result
  Send Rank, class, superclasses and subclasses to GUI
}
End

```

Figure 50. Pseudo code of *Printer*.

4.1.2.5 Graphical User Interface

GUI is the interface in which the user inputs product descriptions in natural language and with the Parser within the *GUI*; *GUI* sends the search keywords to the *QueryBuilder*. After *QueryEngine* and *Printer* processes the query and query result, *GUI* outputs the results back to the user. User also has three options to view the result: first view is the all returned products, second view is the top five products, and the last view is the top 10 products of the result.

In this chapter, the developed applications and their fundamentals as well as keyword creation and injection are explained. In the following chapter, the evaluations of the ontology and test results of the system are presented.

CHAPTER 5

EVALUATION OF THE SYSTEM

5.1. Evaluation of Ontology

Ontology evaluation is defined as the ontology quality according to some criteria in fitting the purpose of the ontology construction (Brank et al., 2005). The main goal of ontology evaluation is to determine if the ontology defines the real world completely, consistently and concisely. Consistency is satisfied by consistent definitions; completeness is satisfied by complete definitions that fully define the real world and serves the explicit information from the definition itself or from the other related definitions; and conciseness is satisfied by the removal of redundancy and unnecessary information in the ontology (Gómez-Pérez et al., 2006).

Ontology verification and ontology validation are two main aspects of ontology evaluation. Ontology verification can be defined as implementing the ontology requirements and construction of the ontology correctly; and ontology validation can be defined as modeling the real world in the ontology and constructing the correct ontology. The assessment of ontology is also done by the users for the ontology definitions and relations constructed in the ontology (Gómez-Pérez et al., 2006).

Ontology evaluations are conducted on several different levels where ontologies can be evaluated by themselves or can be evaluated within some context (Vrandečić, 2009); can be evaluated within an application, called application-based ontology evaluation (Brank et al., 2005), and lastly, in the context of an application and a task, called task-based ontology evaluation (Porzel & Malaka, 2004).

The DL reasoners evaluate three main taxonomic errors of the ontology, which are inconsistency, incompleteness and redundancy (Gómez-Pérez, 1999). The constructed ontology for the thesis is evaluated using two different DL reasoners called HermiT 1.3.8. and FaCT++1.6.2.

The result of the reasoner evaluations by HermiT and FaCT on taxonomic error checking is presented as follows:

- Consistent: The ontology is consistent with no contradiction. Ontology does not have any circulatory errors, partition errors and semantic inconsistency errors
- Complete: All classes and properties are defined in the ontology, no incomplete relation or data property assertion or description.
- No Redundancy: there are any inferred information from the relations, classes and instances of the ontology. All information is defined only once.

Ontology evaluation is also carried on application evaluation level and user evaluation level, as well. The developed application is used for evaluation by searching the classes, relations and data properties, then checking the results, to see whether the results show the correct entities and are in the correct class hierarchy. Query results of the ontology are also checked from the ontology itself using Protégé which is presented in the following sections.

Testing

In this study, the constructed ontology which is based on a part of HS coding system is manually queried for five target keywords *ipek*, *döküntü*, *iplik*, *koza* and *misina* as a test case. In this case, *İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş*

iplikler (perakende satılacak hale getirilmiş); misina (ipek böceği guddesinden elde edilen) is the target product with G.T.İ.P number 5006.00. Since the keywords *ipek*, *döküntü* and *iplik* are inherited keywords, most of the products have these keywords.

The number of the correctly matched keywords always leads the target product into the higher Rank. Here, *ipek* as the most common keyword of ontology is queried first, and the target product is at 19th order as presented in the following Figure.

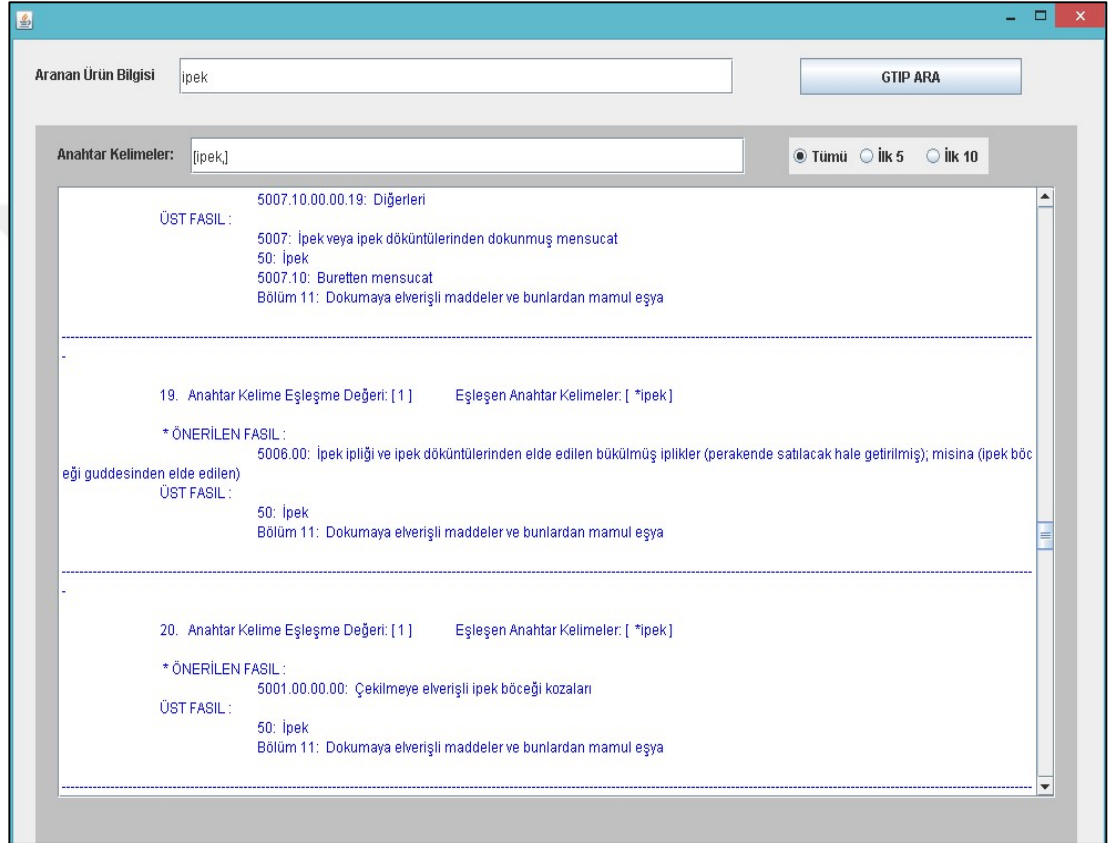


Figure 51. Query result of target product for keyword *ipek*.

Secondly, *ipek* and *döküntü* keywords are queried and the target product is at 17th order as presented in the following figure:

The screenshot shows a search interface with the following elements:

- Aranan Ürün Bilgisi:** ipek döküntü
- GTİP ARA** button
- Anahtar Kelimeler:** [ipek,döküntü,]
- Radio buttons:** Tümü, İlk 5, İlk 10
- Search Results:**
 - 16. Anahtar Kelime Eşleşme Değeri: [2] Eşleşen Anahtar Kelimeler: [*döküntü, *ipek]**
 - * ÖNERİLEN FASIL :** 5007.10.00.00.19: Diğerleri
 - ÜST FASIL :** 5007: İpek veya ipek döküntülerinden dokunmuş mensucat
50: İpek
5007.10: Bütten mensucat
Bölüm 11: Dokumaya elverişli maddeler ve bunlardan mamul eşya
 - 17. Anahtar Kelime Eşleşme Değeri: [2] Eşleşen Anahtar Kelimeler: [*ipek, döküntü]**
 - * ÖNERİLEN FASIL :** 5006.00: İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş iplikler (perakende satılacak hale getirilmiş); misina (ipek böc eği guddesinden elde edilen)
 - ÜST FASIL :** 50: İpek
Bölüm 11: Dokumaya elverişli maddeler ve bunlardan mamul eşya
 - 18. Anahtar Kelime Eşleşme Değeri: [2] Eşleşen Anahtar Kelimeler: [*döküntü, *ipek]**
 - * ÖNERİLEN FASIL :** 5007.20.51.00.00: Ağartılmamış, temizlenmiş veya ağartılmış

Figure 52. Query result of the target product for keywords *ipek* and *döküntü*.

The other inherited keyword *iplik* is queried with the keywords *ipek* and *döküntü*. The target product is 5th order as presented in Figure 53.

The screenshot shows a software interface for searching products. At the top, there is a search bar labeled 'Aranan Ürün Bilgisi' containing the text 'ipek döküntü iplik' and a button labeled 'GTIP ARA'. Below the search bar, there is a section for 'Anahtar Kelimeler:' with the text '[ipek,döküntü,iplik,]' and three radio buttons: 'Tümü' (selected), 'İlk 5', and 'İlk 10'. The main content area displays search results for three different products, each with a list of 'ÖNERİLEN FASIL' and 'ÜST FASIL' items. The first result is for '5003.00.00.00.29: Diğer Döküntüler'. The second result is for '5006.00: İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş iplikler (perakende satılacak hale getirilmiş); misina (ipek böceği guddesinden elde edilen)'. The third result is for '5004.00: İpek ipliği (ipek döküntülerinden elde edilen iplikler hariç) (perakende satılacak hale getirilmemiş)'. Each result also includes 'Anahtar Kelime Eşleşme Değeri' and 'Eşleşen Anahtar Kelimeler'.

Figure 53. Query result of the target product for keywords *ipek*, *döküntü* and *iplik*.

Koza keyword is also queried with the keywords ipek, döküntü and iplik. As a result, the target product is at 3rd order in 39 returned products as presented in the following Figure:

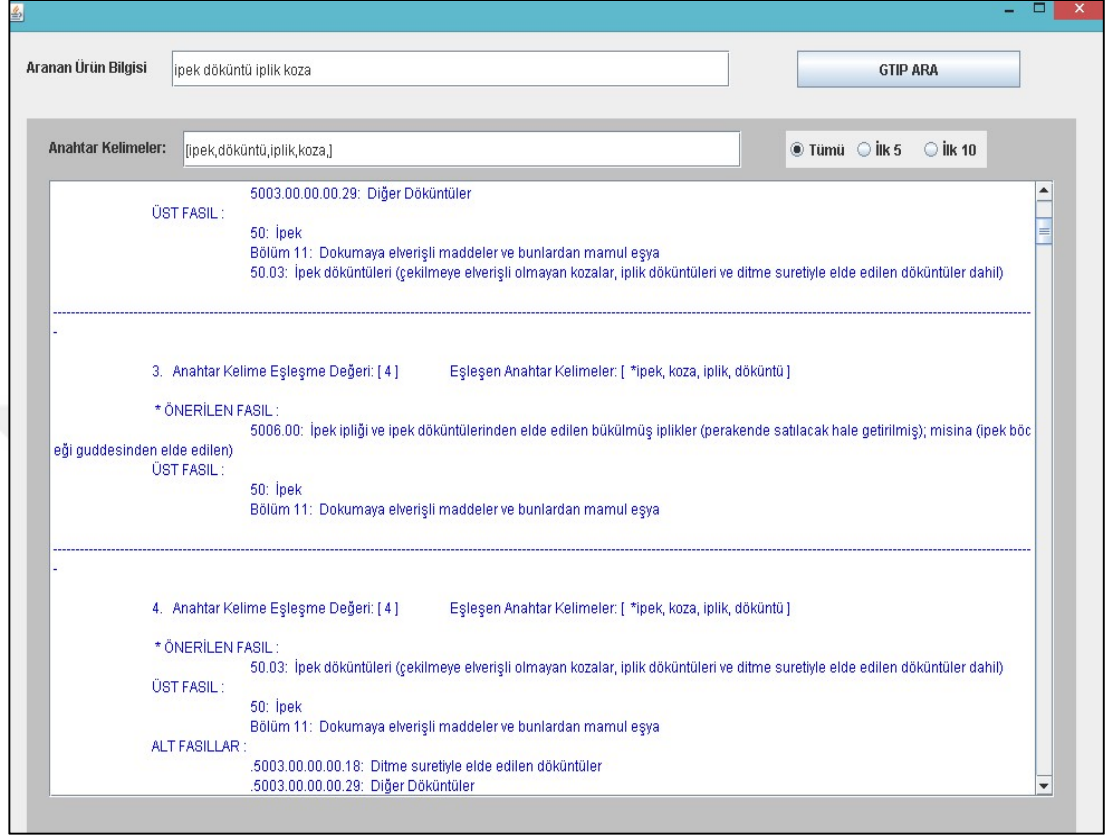


Figure 54. Query result for the target product queried with keywords ipek, döküntü, iplik and koza.

In the following Figure, the result of the query with the keywords *ipek*, *döküntü*, *iplik*, *koza* and *misina* are represented; and the target product is at the first order with the higher *Rank*.

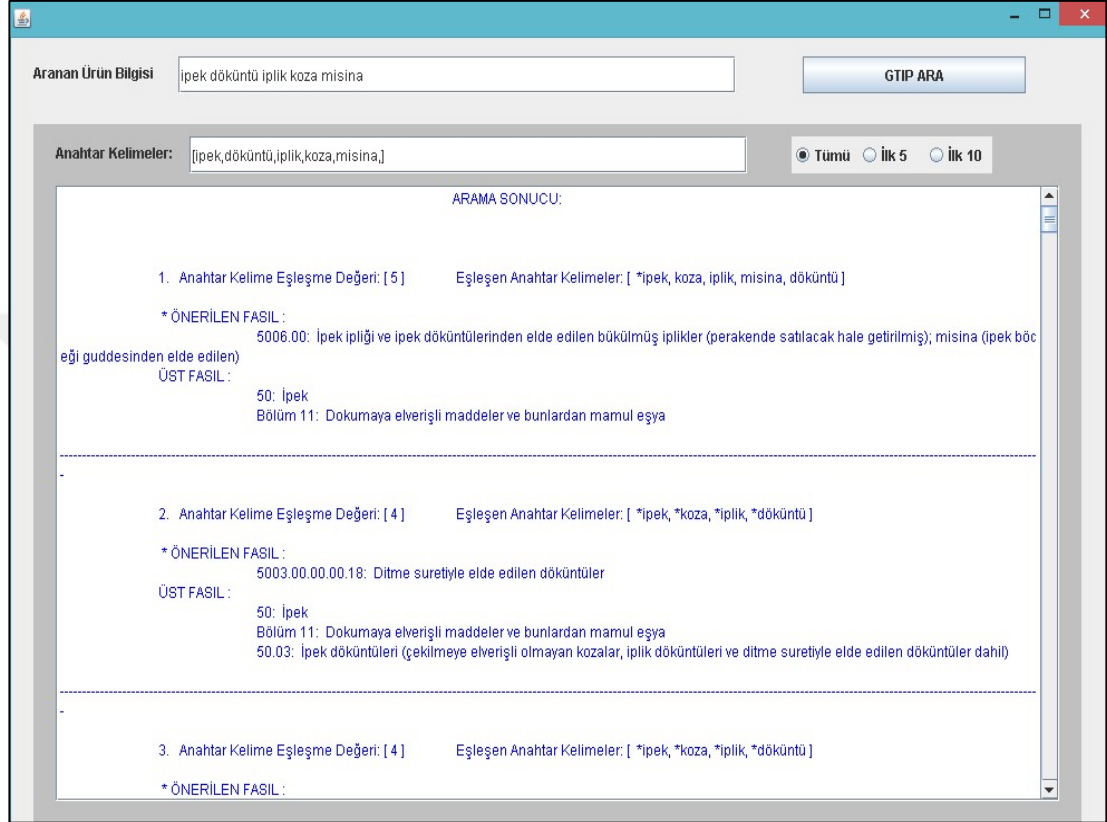


Figure 55. Query result for the target product queried with keywords *ipek*, *döküntü*, *iplik*, *koza* and *misina*.

Additionally, in this study, using *olmayan* and *hariç* keywords or *-siz* suffix, elimination can be done in the query result for the unwanted products. The following test case in the following Figure represents the query result of the products not having *lif* keyword is queried using *-siz* suffix, that means querying with *lifsiz* keyword.

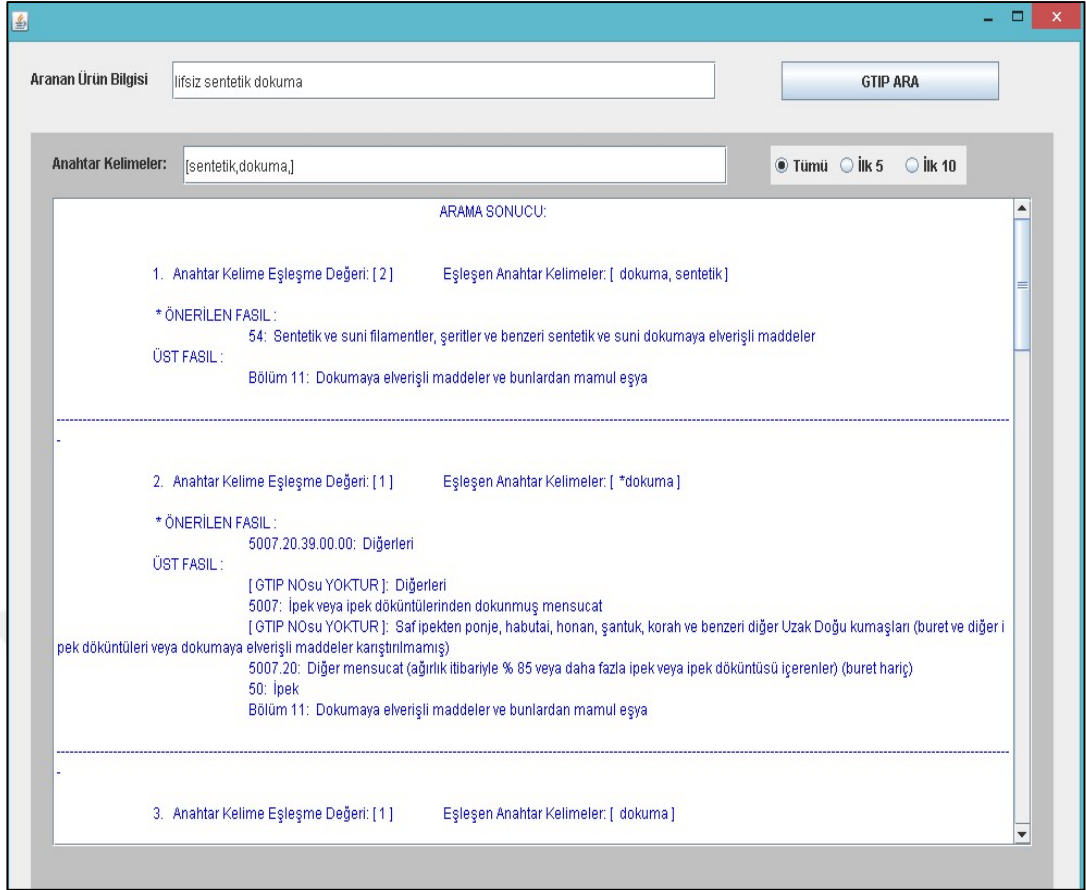


Figure 56. Query result of the product using *-siz* suffix.

In the following Figure, products having keywords *lif* and *dokuma* but not *sentetik* is queried with the keyword *olmayan*. As represented in the Figure only one product matches two keywords *dokuma* and *lif*.

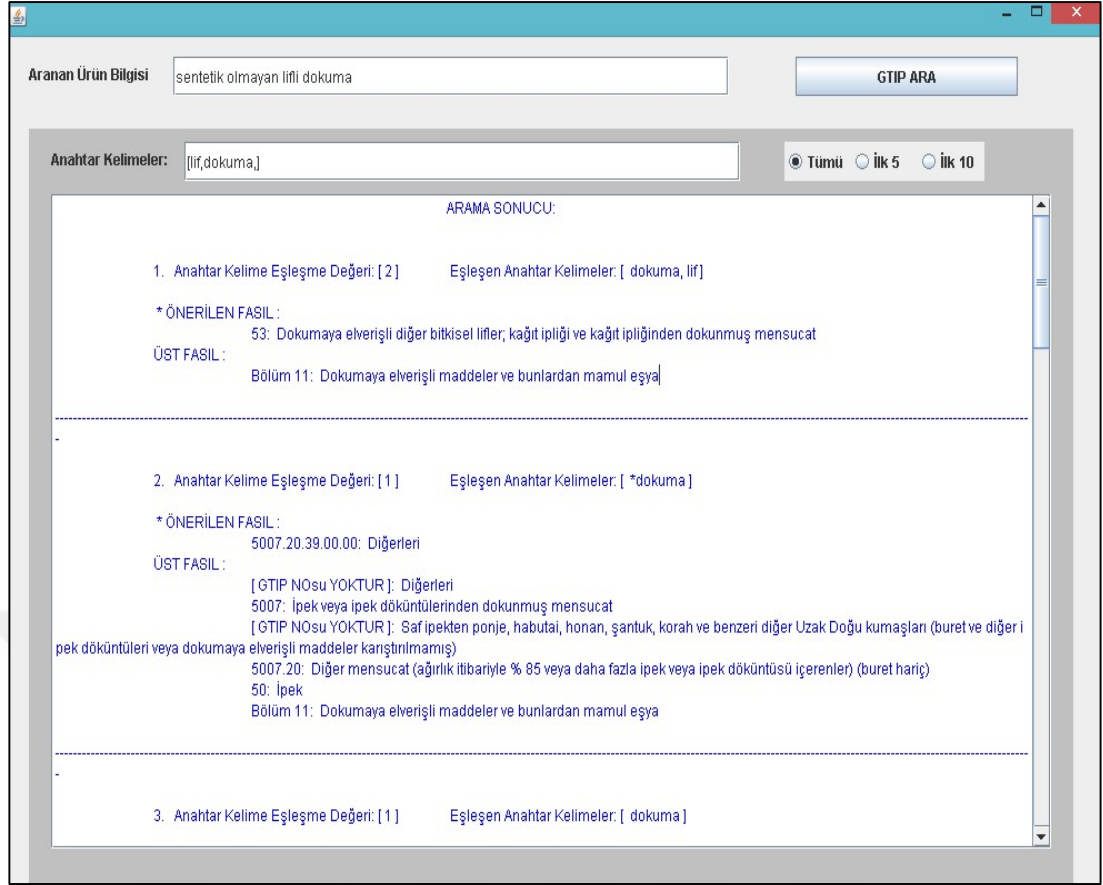


Figure 57. Query result of the product using *olmayan* keyword.

5.2. Query Response Time Comparison

In this study, products in the ontology are queried with different number of keywords, 1 up to 5 keywords. Queries are carried out on the computer with AMD 64X2 2GHz processor, 2GB RAM and Windows 8 operating system. Following Table 9, represents five trials with different number of words and response times.

Table 9. Query response time for different number of keywords in seconds.

Trial No	1 keyword	2 keywords	3 keywords	4 keywords	5 keywords
1	2.71	2.86	2.63	2.79	2.76
2	2.88	2.84	2.51	2.60	2.59
3	2.90	2.57	2.65	2.53	2.79
4	2.87	2.81	2.81	2.76	2.52
5	2.84	2.90	2.85	2.88	2.72

In Appendix A and B, the given ontology metrics indicates the scale of the constructed ontology on subdomain. The following Figure depicts time changes with respect to different number of keywords. Since the queried number of the keywords are different, and even negativity has been tried for five keywords. As seen in the Figure, there exists no significant difference in query response times. Note that, this is not a statistical report but Figure obviously represents that the number of queried keywords has no effect on the query response time of the system.

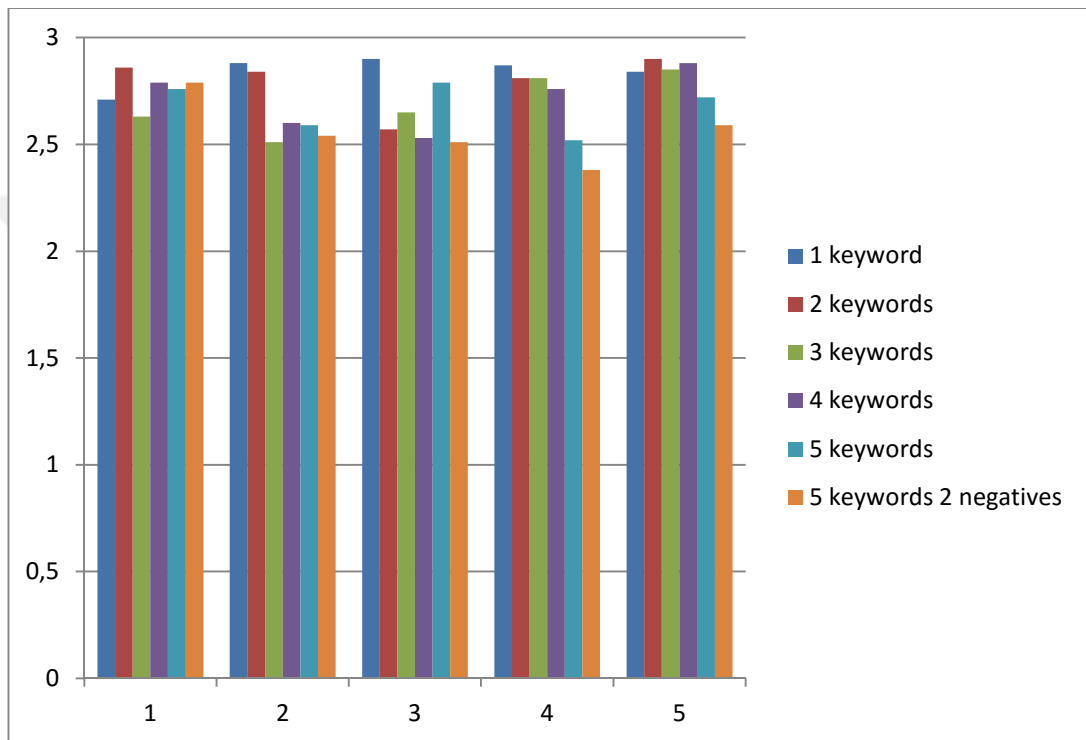


Figure 58. Illustration of keywords query for different number of keywords. Y-axis shows time in seconds and X-axis show number of trials that each trial represents the queries for different number of keywords from 1 up to 5 keywords.

For the evaluation of the system, 100 queries have been tested, 25 with 1 word keyword, 25 with 3 keywords, 25 with 5 keywords, and the last 25 with negativity components such as a negativity suffix *-me/ma* or words that imply negativity *olmayan, hariç* etc. For all categories the results of the query are ranked, and the rank of the correct G.T.İ.P. number is checked to see if it is found in 1st ranked result, in

the first 5 result or in the first 10 result. The table below gives the summary of the results.

Table 10. Query result for 1 keywords search.

Query	1 st ranked	First 5 ranked	First 10 ranked
1 keyword	5	11	14

Table 11. Query result for 3 keywords search.

Query	1 st ranked	First 5 ranked	First 10 ranked
3 keywords	11	15	15

Table 12. Query result for 5 keywords search.

Query	1 st ranked	First 5 ranked	First 10 ranked
5 keywords	16	23	23

Table 13. Query result for 3 keywords with negative component search.

Query	1 st ranked	First 5 ranked	First 10 ranked
3 keywords with negative	5	16	21

As seen in the above table, the accuracy rate increased when the number of keywords is higher. And also Queries with the specific keywords to the products increases the rank order. The following table presents the results for all of the 100 queries. In order to have more accurate results, synonyms and number of keywords for the products must be increased. Queried sentences and their corresponding results is given in Appendix C, D, E and F.

Table 13. Query result for 100 queries.

Query	1 st ranked	First 5 ranked	First 10 ranked
100 queries	37	65	73

CHAPTER 6

CONCLUSION AND DISCUSSION

In this thesis, an ontology based on the Turkish HS coding system is constructed. Turkish HS coding system is a part of customs international HS coding system where some part of the codes are identical between nations but the rest differs with respect to countries. Originality of this thesis comes from the special integration of a Turkish NLP tool with ontology construction and searching modules. Although there are some examples in previous research that also integrate such components, this is the first approach for the Turkish language.

In the first phase of thesis, HS coding system and its specific structure is analyzed and keywords of the products and their groups are determined using Turkish NLP. Deciding about the NLP tool and the level of analysis are important parts of this thesis, such that using more levels does not increase the effectiveness or efficiency. For example, rather than using only morphological analysis, using discourse analysis additionally may provide better information about the given input but it will also increase the cost of the application where 5000 different entities need to be represented and queried efficiently.

Integration of a Turkish NLP tool and an ontology was relatively easy part of the study where Java provides useful libraries for both. However, querying the ontology and reducing the query costs were the most complicated part of the study. There are number of ways to construct an ontology and querying it. Nevertheless, it is not always useful and cheap, so there are plenty of hours and thousands of lines of codes produced to reduce this cost.

As a result of this thesis, a new perspective for ontology design especially for Turkish language is presented. Querying of this ontology considers the morphological information of input sentences, extracts meaningful interpretation from the sentence, supports a query based on this information and presents the result via the user interface.

Last but not least, before the result is presented to the user, morphological information is reused by the application for some additional comparisons. Since Protégé does not provide the *NOT* operator for querying, an algorithm is developed which uses morphological information and eliminates related results. This is also a novel approach that supports the querying quality for the user where redundant outputs are eliminated.

Reliability and validity analysis of the developed ontology have been performed. Since all the manual search results and the results of the application are the same, this study satisfies its goal which is querying in ontology with the natural language where hopefully will enable non-expert users to find the G.T.İ.P. number of any product, given its Turkish characteristics.

At the end, this study provides a flexible and efficient querying method for particularly the Turkish HS coding system. Since the ontology domain can easily be changed, this structure can be used for other domains as well. In order to use this approach, designers must be aware of some restrictions.

The first restriction is the target domain to be queried must have a similar class hierarchy. Recall that in Turkish HS coding system; each division has its own special

family and under it, each entity has its own unique name, in other words, each entity in the domain must be unique and all the classes must be disjoint.

The second restriction belongs to the language dependency; to be clearer, language can be same but the used words in the domain may differ. Note that, we highlight the *Parser* module at the Chapter 3 that identifies some special words such as *hariç*, *dışında* and also some affixes such as *siz*, *-me/ma*. Anyone who may use this approach needs to consider these special words and affixes used in Turkish language which changes the meaning of the word in question.

In this study, the keyword creation is done by analyzing the sample domain data. Since the system's accuracy and search quality depends on the keywords, the constraints of the system is dependent on the number and representational power of these keywords.

Another constraint related to this study can be seen in the future. If the number of the products increases enormously, then system response time may increase accordingly. So in such a scenario, multiple ontology structure can be a solution where chapters of products can be represented in multiple ontologies.

To conclude, this study provides a new and promising tool for Turkish querying systems. Integration of the NLP tool with the ontology, and creation of such a querying system may become a new trend in the research on Turkish language. This approach may lead to researchers to use such a structure to improve the quality of Turkish searching mechanism.

This study supports users to describe the entities with their natural language as they do not need to be aware of the entities description in the domain. In this aspect there is only one limitation for the system that the keywords injected into the ontology must cover the description of the target entities in natural language.

To be more specific, during the ontology construction expert or non-expert users do not need to know the HS code of the target product or its name while using this new system, only its basic description is required. For example, searching for a product

named *ipek* can be queried simply with *parlak ince kumaş*. If any of these words are injected to the system as keywords for the product *ipek*, the system will respond with the name of the entity *ipek* and its unique HS code. Additionally, system will also bring other similar entities which have some of these keywords. Note that number of keywords correspondence to the input sentence will be represented with a ranking system where the more matching input with the keywords will result in a higher rank. Hopefully, any user, expert or non-expert, will be able to search for any products' *G.T.I.P* easily, effectively and efficiently in Turkish from the developed HS code ontology.

Future Work

As future work, the system can be tested by customs officers to receive feedback and to improve the system. Efficiency of querying can be increased with more keywords related to the target products. In order to define more expressive and frequent keywords, a survey can be done with respect to the domain. Designer can also process the corpus of the target domain that can provide useful keyword candidates for the system. Moreover, after the application of this system, all the queries and the results can be recorded and used for new keyword identification. Additionally, integrating a synonym dictionary module into the system may be useful. For example, if the input sentence does not include any word which is being represented in the ontology, the parsed sentence elements can be replaced with the synonyms and queried again. It will obviously bring additional cost but inevitably will increase the efficiency of the application. At the end, Turkish semantic search engines based on ontology in different domain could be constructed based on a similar Turkish NLP tool.

REFERENCES

- Abburu, S. (2012). A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, 57(17).
- Akın, A. A., & Akın, M. D. (2007). Zemberek, an open source NLP framework for Turkic languages. *Structure*, 10, 1-5.
- Alatrish, E. S. (2013). Comparison Some of Ontology. *Journal of Management Information Systems*, 8(2), 018-024.
- Amasyalı, M. F. (2011). Kavramlar Arası Anlamsal İlişkilerin Türkçe Sözlük Tanımları Kullanılarak Otomatik Olarak Çıkarılması/matic Extraction of Semantic Relationships using Turkish Dictionary Definitions. *EMO Bilimsel Dergi*, 1(1), 1-14.
- Antoniou, G., & Van Harmelen, F. (2004). Web ontology language: Owl. In *Handbook on ontologies* (pp. 67-92). Springer Berlin Heidelberg.
- Arpírez, J. C., Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2001, October). WebODE: a scalable workbench for ontological engineering. In *Proceedings of the 1st international conference on Knowledge capture* (pp. 6-13). ACM.
- Baader, F. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.
- Baader, F., Horrocks, I., & Sattler, U. (2001). Description Logics for the Semantic Web.
- Berners-Lee, T., & Fischetti, M. (1999). *Weaving the Web*. HarperSanFrancisco. Chapter 12. ISBN 978-0-06-251587-2.
- Bilgin, O., Çetinoğlu, Ö., & Oflazer, K. (2004). Building a wordnet for Turkish. *Romanian Journal of Information Science and Technology*, 7(1-2), 163-172.
- Blomqvist, E. (2005, October). Fully automatic construction of enterprise ontologies using design patterns: Initial method and first experiences. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (pp. 1314-1329). Springer Berlin Heidelberg.
- Blomqvist, E., Öhgren, A., & Sandkuhl, K. (2006, May). Ontology Construction in an Enterprise Context: Comparing and Evaluating Two Approaches. In *ICEIS (3)* (pp. 86-93).

- Bock, J., Haase, P., Ji, Q., & Volz, R. (2008, June). Benchmarking OWL reasoners. In *ARea2008-Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*. Tenerife.
- Brank, J., Grobelnik, M., & Mladenic, D. (2005, October). A survey of ontology evaluation techniques. In *Proceedings of the conference on data mining and data warehouses (SiKDD 2005)* (pp. 166-170).
- Bruckschen, M., Northfleet, C., Silva, D. M., Bridi, P., Granada, R., Vieira, R., ... & Sander, T. (2010). Named entity recognition in the legal domain for ontology population. In *Workshop Programme* (p. 16).
- Buitelaar, P., Cimiano, P., & Magnini, B. (2005). Ontology learning from text: An overview. *Ontology learning from text: Methods, evaluation and applications*, 123, 3-12.
- Carki, K., Geutner, P., & Schultz, T. (2000). Turkish LVCSR: towards better speech recognition for agglutinative languages. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on* (Vol. 3, pp. 1563-1566). IEEE.
- Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004, May). Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (pp. 74-83). ACM.
- Choi, C., Cho, M., Kang, E. Y., & Kim, P. (2006, February). Travel ontology for recommendation system based on semantic web. In *2006 8th International Conference Advanced Communication Technology* (Vol. 1, pp. 624-627). IEEE.
- Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1), 51-89.
- Copstake, A., Corbett, P., Murray-Rust, P., Rupp, C. J., Siddharthan, A., Teufel, S., & Waldron, B. (2006). An architecture for language processing for scientific texts. In *Proceedings of the UK e-Science All Hands Meeting 2006*.
- Cöltekin, C. (2010, May). A Freely Available Morphological Analyzer for Turkish. In *LREC*.
- Daille, B., Fabre, C., & Sébillot, P. (2002). Applications of computational morphology. *Many morphologies*, 210-234.
- DataONE, (2016) Protégé. Retrieved from, <https://www.dataone.org/software-tools/protege>

Dautov, R., Paraskakis, I., & Kourtesis, D. (2012). An ontology-driven approach to self-management in cloud application platforms. In *Proceedings of the 7th South East European Doctoral Student Conference (DSC 2012)* (pp. 539-550).

Domingue, J., Motta, E., & Garcia, O. C. (1999). Knowledge modelling in webonto and ocml: A user guide.

Eryigit, G. (2014, April). ITU Turkish NLP Web Service. In *EACL* (pp. 1-4).

Eryigit, G., & Adali, E. (2004, February). An affix stripping morphological analyzer for Turkish. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications, Innsbruck, Austria* (pp. 299-304).

Euzenat, J., Polleres, A., & Scharffe, F. (2008, March). Processing ontology alignments with SPARQL. In *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on* (pp. 913-917). IEEE.

Fensel, D. (2001). Ontologies. In *Ontologies* (pp. 11-18). Springer Berlin Heidelberg.

Gandon, F. (2002). *Distributed Artificial Intelligence and Knowledge Management: ontologies and multi-agent systems for a corporate semantic web* (Doctoral dissertation, Université Nice Sophia Antipolis).

Gardent, C., & Webber, B. (1998). Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop, Philadelphia*.

Gentner, D., van Harmelen, F., Hitzler, P., Janowicz, K., & Kuhnberger, K. U. (2012). Cognitive approaches for the semantic web.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HerMiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3), 245-269.

Goh, C. H. (1996). *Representing and reasoning about semantic conflicts in heterogeneous information systems* (Doctoral dissertation, Massachusetts Institute of Technology).

Gómez-Pérez, A. (1999). Evaluation of taxonomic knowledge in ontologies and knowledge bases.

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2006). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media.

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). The Most Outstanding Ontologies. *Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*, 47-106.

- Gruber, T. R. (1992). *Ontolingua: A mechanism to support portable ontologies* (Vol. 27). Stanford: Stanford University, Knowledge Systems Laboratory.
- Hepp, Martin., Radingere, A., (n.d.), In *eClassOWL*. Retrieved, from <http://www.heppnetz.de/projects/eclassowl/>
- Heuristic. (n.d.).(para.1). In *WCO in brief*. Retrieved from <http://www.wcoomd.org/en/about-us/what-is-the-wco.aspx>
- Heuristic. (n.d.). In *Pizza Ontology*. Retrieved, from <http://protege.stanford.edu/ontologies/pizza/pizza.owl>
- Horrige, M., & Bechhofer, S. (2009, October). The OWL API: a Java API for working with OWL 2 ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions-Volume 529* (pp. 49-58). CEUR-WS. org.
- Horrige, M., Tsarkov, D., & Redmond, T. (2006, November). Supporting Early Adoption of OWL 1.1 with Protege-OWL and FaCT++. In *OWLED*.
- Horrocks, I., Parsia, B., Patel-Schneider, P., & Hendler, J. (2005, September). Semantic web architecture: Stack or two towers?. In *International Workshop on Principles and Practice of Semantic Web Reasoning* (pp. 37-41). Springer Berlin Heidelberg.
- ITU, ITU Turkish NLP Pipeline. (n.d.). Retrieved from, <http://tools.nlp.itu.edu.tr>.
- Jackson, P., & Moulinier, I. (2007). *Natural language processing for online applications: Text retrieval, extraction and categorization* (Vol. 5). John Benjamins Publishing.
- Jaimes, A., & Smith, J. R. (2003, July). Semi-automatic, data-driven construction of multimedia ontologies. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference On* (Vol. 1, pp. I-781). IEEE.
- Jansen, L. (2008). Categories: the top-level ontology. *Applied ontology*, 173-196.
- Jarrar, M., (2015). Introduction to NLP lecture notes. Retrieved from http://www.jarrar.info/courses/AAI/Jarrar.LectureNotes.NLP_P1_Introduction.pdf
- Kapoor, B., & Sharma, S. (2010). A comparative study ontology building tools for semantic web applications. *International journal of Web & Semantic Technology (IJWesT)*, 1(3).
- Katz, J. J., & Fodor, J. A. (1963). The structure of a semantic theory. *language*, 39(2), 170-210.

Khondoker, M. R., & Mueller, P. (2010, March). Comparing ontology development tools based on an online survey. World Congress on Engineering 2010 (WCE 2010), London, UK.

Knublauch, H. (2010, June 30). Protege-OWL API Programmer's Guide. Retrieved from, http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide

Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004, November). The Protégé OWL plugin: An open development environment for semantic web applications. In *International Semantic Web Conference* (pp. 229-243). Springer Berlin Heidelberg.

Laporte, É. (2005). Symbolic natural language processing. *Applied Combinatorics on Words*, 164-209.

Lapp, H. (2016, January 14). Reasoners, OWL API Support, papers about the OWL API. Retrieved from, <https://github.com/owlcs/owlapi/wiki/Reasoners,-OWL-API-Support,-papers-about-the-OWL-API>

Lee, T., Lee, I. H., Lee, S., Lee, S. G., Kim, D., Chun, J., ... & Shim, J. (2006). Building an operational product ontology system. *Electronic Commerce Research and Applications*, 5(1), 16-28.

Lenzerini, M., Milano, D., & Poggi, A. (2004). Ontology representation & reasoning. *Universit di Roma La Sapienza, Roma, Italy, Tech. Rep. NoE InterOp (IST-508011)*.

Liddy, E. D. (2001). Natural language processing.

Liddy, E. D., Paik, W., McKenna, M. E., Weiner, M. L., Edmund, S. Y., Diamond, T.G., ... & Snyder, D. L. (2000). US Patent No.6,026,388. Washington, DC: U.S. Patent and Trademark Office.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing* (Vol. 999). Cambridge: MIT press.

Mayer, R., Plank, C., Bohner, A., Kollarits, S., Corsini, A., Ronchetti, F., ... & Tosoni, D. (2008). MONITOR: hazard monitoring for risk assessment and risk communication. *Georisk*, 2(4), 195-222.

McBride, B. (2001, May). Jena: Implementing the rdf model and syntax specification. In *Proceedings of the Second International Conference on Semantic Web-Volume 40* (pp. 23-28). CEUR-WS. org.

Miller, E. (1998). An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology*, 25(1), 15-19.

Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11), 39-41.

Motik, B., Patel-Schneider, P. F., Parsia, B., Bock, C., Fokoue, A., Haase, P., ... & Smith, M. (2009). OWL 2 web ontology language: Structural specification and functional-style syntax. *W3C recommendation*, 27(65), 159.

Möller, M., & Sintek, M. (2007). A Generic Framework for Semantic Medical Image Retrieval. *KAMC*, 253.

Nguyen, V. (2011). *Ontologies and Information Systems: A Literature Survey* (No. DSTO-TN-1002). DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA).

Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: A guide to creating your first ontology*.

O'connor, M., Knublauch, H., Tu, S., & Musen, M. (2005). Writing rules for the semantic web using SWRL and Jess. *Protégé With Rules WS, Madrid*.

Oflazer, K. (1994). Two-level description of Turkish morphology. *Literary and linguistic computing*, 9(2), 137-148.

Orhan, Z., Pehlivan, İ., Uslan, V., & Önder, P. (2011). Automated extraction of semantic word relations in turkish lexicon. *Mathematical and Computational Applications*, 16(1), 13-21.

Ovchinnikova, E. (2012). *Integration of world knowledge for natural language understanding* (Vol. 3). Springer Science & Business Media.

Pan, J. Z., & Thomas, E. (2007, July). Approximating owl-dl ontologies. In *Proceedings of the National Conference on Artificial Intelligence* (Vol. 22, No. 2, p. 1434). Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Parsia, B., & Sirin, E. (2004, November). Pellet: An owl dl reasoner. In *Third International Semantic Web Conference-Poster* (Vol. 18).

Peim, M., Franconi, E., Paton, N. W., & Goble, C. A. (2002). Query processing with description logic ontologies over object-wrapped databases. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on* (pp. 27-36). IEEE.

Porzel, R., & Malaka, R. (2004, August). A task-based approach for ontology evaluation. In *ECAI Workshop on Ontology Learning and Population, Valencia, Spain*.

Pretorius, L., & Bosch, S. E. (2003). Enabling computer interaction in the indigenous languages of South Africa: The central role of computational morphology. *interactions*, 10(2), 56-63.

- Pustejovsky, J. (1991). The generative lexicon. *Computational linguistics*, 17(4), 409-441.
- Reilly, R. G., & Sharkey, N. E. (1992). *Connectionist approaches to natural language processing*. Lawrence Erlbaum Associates, Inc.
- Reshamwala, A., Mishra, D., & Pawar, P. (2013). Review on natural language processing. *IRACST Engineering Science and Technology: An International Journal (ESTIJ)*, 3, 113-116.
- Sahin, M., Sulubacak, U., & Eryigit, G. (2013, October). Redefinition of Turkish morphology using flag diacritics. In *Proceedings of The Tenth Symposium on Natural Language Processing (SNLP-2013), Phuket, Thailand, October*.
- Sak, H., Güngör, T., & Saraçlar, M. (2011). Resources for Turkish morphological processing. *Language resources and evaluation*, 45(2), 249-261.
- Sak, H., Güngör, T., & Saraçlar, M. (2008). Turkish language resources: Morphological parser, morphological disambiguator and web corpus. In *Advances in natural language processing* (pp. 417-427). Springer Berlin Heidelberg.
- Saripalle, R. K., Demurjian, S. A., De la Rosa Algarín, A., & Blechner, M. (2013). A software modeling approach to ontology design via extensions to ODM and OWL. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 9(2), 62-97.
- Semantik MediaWiki. (2013, March 21). Protégé Plugins Library. Retrieved from, http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library
- Sharkey, N. E. (1992). *Connectionist natural language processing: readings from Connection science*. Intellect Books.
- Singh, A., & Anand, P. (2013). State of Art in Ontology Development Tools. *International Journal*, 2(7).
- SOURCEFORGE.NET. (2016). The OWL API. Retrieved from, <http://owlapi.sourceforge.net/>
- Sowa F.J., Top-Level Categories. (2001, December 04). Retrieved, from <http://users.bestweb.net/~sowa/ontology/toplevel.htm>
- Stanford Center for Biomedical Informatics Research, (2016) Protégé. Retrieved from, <http://protege.stanford.edu/>
- Subhashini, R., & Akilandeswari, J. (2011). A survey on ontology construction methodologies. *International Journal of Enterprise Computing and Business Systems*, 1(1), 60-72.

Swaak, J., & De Jong, T. (2001). Discovery simulations and the assessment of intuitive knowledge. *Journal of Computer Assisted Learning*, 17(3), 284-294.

Swartout, B., Patil, R., Knight, K., & Russ, T. (1996, November). Toward distributed use of large-scale ontologies. In *Proc. of the Tenth Workshop on Knowledge Acquisition for Knowledge-Based Systems* (pp. 138-148).

Şahin, M. (2014). ITUMorph, a more accurate and faster wide coverage morphological analyzer for Turkish. Master's thesis, Istanbul Technical University.

Tallerman, M. (2014). *Understanding syntax*. Routledge.

The University of Manchester. (2016). Old List of Reasoners. Retrieved from <http://owl.cs.manchester.ac.uk/tools/old-list-of-reasoners/>

The Apache Software Foundation. (2016). Jena Ontology API. Retrieved from, <https://jena.apache.org/documentation/ontology/#general-concepts>

Tsarkov, D. (n.d.) Incremental and Persistent Reasoning in FaCT+.

Tsarkov, D., & Horrocks, I. (2006, August). FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Reasoning* (pp. 292-297). Springer Berlin Heidelberg.

Tudorache, T., Noy, N. F., Tu, S., & Musen, M. A. (2008, October). Supporting collaborative ontology development in Protégé. In *International Semantic Web Conference* (pp. 17-32). Springer Berlin Heidelberg.

Van Der Vet, P. E., & Mars, N. J. (1998). Bottom-up construction of ontologies. *IEEE Transactions on Knowledge and data Engineering*, 10(4), 513-526.

Vrandečić, D. (2009). Ontology evaluation. In *Handbook on Ontologies* (pp. 293-313). Springer Berlin Heidelberg.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hübner, S. (2001, August). Ontology-based integration of information-a survey of existing approaches. In *IJCAI-01 workshop: ontologies and information sharing* (Vol. 2001, pp. 108-117).

Witte, R., Khamis, N., & Rilling, J. (2010, May). Flexible Ontology Population from Text: The OwlExporter. In *LREC* (Vol. 2010, pp. 3845-3850).

W3C Semantic Web Activity. (2013, December 11). Retrieved, from <https://www.w3.org/2001/sw/>

W3C Semantic Web Standards. (2014, January 28). Retrieved, from https://www.w3.org/2001/sw/wiki/Main_Page

Yavuz, H., & BALPINAR, Z. (2011). Turkish Phonology and Morphology.

Zang, B., Li, Y., Xie, W., Chen, Z., Tsai, C. F., & Laing, C. (2008). An ontological engineering approach for automating inspection and quarantine at airports. *Journal of Computer and System Sciences*, 74(2), 196-210.

Zhang, Z., & Miller, J. A. (2005). *Ontology query languages for the semantic web: A performance evaluation* (Doctoral dissertation, Master's thesis, University of Georgia).

Zemberek Contributors. (2013). Retrieved from, <https://github.com/ahmetaa/zemberek-nlp/blob/master/contributors.txt>.



APPENDICES

APPENDIX A

Ontology metrics:

Metrics	
Axiom	730
Logical axiom count	550
Class count	54
Object property count	4
Data property count	3
Individual count	51
DL expressivity	SO(D)

Class Axioms:

Class axioms	
SubClassOf axioms count	86
EquivalentClasses axioms count	0
DisjointClasses axioms count	9
GCI count	0
Hidden GCI Count	0

Object property axioms:

Object property axioms	
SubObjectPropertyOf axioms count	0
EquivalentObjectProperties axioms count	0
InverseObjectProperties axioms count	2
DisjointObjectProperties axioms count	0
FunctionalObjectProperty axioms count	0
InverseFunctionalObjectProperty axioms count	0
TransitiveObjectProperty axioms count	4
SymmetricObjectProperty axioms count	0
AsymmetricObjectProperty axioms count	0
ReflexiveObjectProperty axioms count	0
IrreflexiveObjectProperty axioms count	0
ObjectPropertyDomain axioms count	4
ObjectPropertyRange axioms count	4
SubPropertyChainOf axioms count	0

APPENDIX B

Data property axioms:

Data property axioms	
SubDataPropertyOf axioms count	0
EquivalentDataProperties axioms count	0
DisjointDataProperties axioms count	0
FunctionalDataProperty axioms count	0
DataPropertyDomain axioms count	0
DataPropertyRange axioms count	3

Individuals' axioms:

Individual axioms	
ClassAssertion axioms count	101
ObjectPropertyAssertion axioms count	63
DataPropertyAssertion axioms count	272
NegativeObjectPropertyAssertion axioms count	0
NegativeDataPropertyAssertion axioms count	0
SameIndividual axioms count	0
DifferentIndividuals axioms count	2

Annotation axioms:

Annotation axioms	
AnnotationAssertion axioms count	68
AnnotationPropertyDomain axioms count	0
AnnotationPropertyRangeOf axioms count	0

APPENDIX C

<i>1 keyword Query Results</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
1	iplik	5006.00: İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş iplikler (perakende satılacak hale getirilmiş); misina (ipek böceği guddesinden elde edilen)	iplik	12
2	kumaş	5007.10.00.00.11: El tezgahlarında dokunanlar	kumaş	24
3	örgü	60: Örme Mensucat	örgü	1
4	dokuma	58: Özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler	dokuma	10
5	döküntü	50.03: İpek döküntüleri (çekilmeye elverişli olmayan kozalar, iplik döküntüleri ve ditme suretiyle elde edilen döküntüler dahil)	döküntü	24
6	ipek	50: İpek	ipek	5
7	koza	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	koza	6
8	bez	5007.20.31.00.00: Bez ayağı (düz dokunmuş)	bez	2
9	kıyafet	63: Dokumaya elverişli maddelerden diğer hazır eşya; takımlar; kullanılmış giyim eşyası ve dokumaya elverişli maddelerden kullanılmış eşya; paçavralar	kıyafet	1
10	döküntü	5003.00.00.00.18: Ditme suretiyle elde edilen döküntüler	döküntü	4
11	naylon	54: Sentetik ve suni filamentler, şeritler ve benzeri sentetik ve suni dokumaya elverişli maddeler	naylon	2
12	ipek	5005.00: İpek döküntülerinden elde edilen iplikler (perakende satılacak hale getirilmemiş)	ipek	35
13	ipek	50.03: İpek döküntüleri (çekilmeye elverişli olmayan kozalar, iplik döküntüleri ve ditme suretiyle elde edilen döküntüler dahil)	ipek	27

<i>1 keyword Query Results (Continued)</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
14	iplik	5006.00: İpek ipliği ve ipek döküntülerinden elde edilen bükülmüş iplikler (perakende satılacak hale getirilmiş); misina (ipek böceği guddesinden elde edilen)	iplik	11
15	mensucat	8: Özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler	mensucat	29
16	buret	5007.10: Burettten mensucat	buret	2
17	krep	[GTIP NOSu YOKTUR]: Krepler	krep	1
18	tüy	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	tüy	1
19	iplik	5007.90.50.00.00: Farklı renkteki ipliklerden	iplik	12
20	iplik	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	iplik	10
21	yün	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	yün	1
22	dokuma	5007.20.31.00.00: Bez ayağı (düz dokunmuş)	dokuma	11
23	tekstil	5007: İpek veya ipek döküntülerinden dokunmuş mensucat	tekstil	4
24	ipek	5005.00: İpek döküntülerinden elde edilen iplikler (perakende satılacak hale getirilmemiş)	İpek	32
25	ipek	5007.10: Burettten mensucat	ipek	23

APPENDIX D

<i>3 keywords Query Results</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
1	dokuma duvar halısı	58: Özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler	halı, duvar, dokuma	1
2	batik kumaş tekstil	5007.90.90.00.00: Baskılı	tekstil, kumaş, batik	1
3	değişik renklerde kumaşlar	5007.90.50.00.00: Farklı renkteki	kumaş, renk, değişik	1
4	burettten yapılmış tekstil	5007.10: Burettten mensucat	tekstil, burettten	2
5	pamuk ipliğinden eşya	52: Pamuk	pamuk	1
6	keçi tüyü kılı	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	tüy, kıl	1
7	dokunmuş ipek dantel	58: Özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler	dantel, dokun	25
8	kullanılmış kıyafet giysi	63: Dokumaya elverişli maddelerden diğer hazır eşya; takımlar; kullanılmış giyim eşyası ve dokumaya elverişli maddelerden kullanılmış eşya; paçavralar	kıyafet, giysi, kullan	1
9	kullanılmamış örülmemiş kıyafet	62: Örülmemiş giyim eşyası ve aksesuarı	kıyafet, örülme	1
10	ipek kumaşlar tekstil	5007: İpek veya ipek döküntülerinden dokunmuş mensucat	tekstil, kumaş	4

3 keyword Query Results (Continued)				
No	Query Sentence	Target Product	Matched Keywords	Rank
11	baskılı renkli kumaş	5007.20.71.00.00: Baskılı	kumaş, baskı	5
12	ham saf ipek	5002.00.00.00.00: Ham ipek (bükülmemiş)	ipek, ham	2
13	bükülmemiş ipek lifleri	5002.00.00.00.00: Ham ipek (bükülmemiş)	ipek, bükülme	1
14	kimyasal maddeden lif	55: Sentetik ve suni devamsız lifler	lif, kimyasal	1
15	korah veya şantuk	[GTIP NOsu YOKTUR]: Saf ipekten ponje, habutai, honan, şantuk, korah ve benzeri diğer Uzak Doğu kumaşları (buret ve diğer ipek döküntüleri veya dokumaya elverişli maddeler karıştırılmamış)	korah, şantuk	3
16	düz dokuma kumaşlar	5007.20.31.00.00: Bez ayağı (düz dokunmuş)	kumaş, dokuma, düz	1
17	döküntü karıştırılmamış ipek	[GTIP NOsu YOKTUR]: Saf ipekten ponje, habutai, honan, şantuk, korah ve benzeri diğer Uzak Doğu kumaşları (buret ve diğer ipek döküntüleri veya dokumaya elverişli maddeler karıştırılmamış)	döküntü, ipek, karıştırılma	15
18	örgü aksesuar süs	61: Örne giyim eşyası ve aksesuarı	aksesuar, örgü	1
19	işlenmemiş ipek kozası	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	ipek, koza	4
20	yumuşak parlak kumaş	50: İpek	parlak, yumuşak	29
21	dokunmuş kumaş ince	5007.20.51.00.00: Ağartılmamış, temizlenmiş veya ağartılmış	dokun, kumaş, ince	13
22	ipek böceği hayvan	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	ipek, hayvan, böcek	2
23	ipek çöpü kırık	50.03: İpek döküntüleri (çekilmeye elverişli olmayan kozalar, iplik döküntüleri ve ditme suretiyle elde edilen döküntüler dahil)	ipek	27

<i>3 keyword Query Results (Continued)</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
24	parça ipekten yapılmış	5007.20:Diğer mensucat(ağırlık itibariyle % 85 veya daha fazla ipek veya ipek döküntüsü içerenler)(buret hariç)	ipek	12
25	yıllanmış pamuktan kumaş	52: Pamuk	pamuk	17

APPENDIX E

<i>5 keywords Query Results</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
1	tığ işi yumuşak örtü süs	60: Örme Mensucat	tığ, yumuşak	1
2	beyazlatılmış ağartılmış krep kumaştan eşya	5007.20.11.00.00: Ağartılmamış, temizlenmiş veya ağartılmış	kumaş, krep, ağar	1
3	şiş ile örülmüş kıyafet giysi	61: Örme giyim eşyası ve aksesuarı	kıyafet, giysi, şiş	1
4	beyazlatılmış yıkanmış ipek ipliğinden eşyalar	5007.90.10.00.00: Ağartılmamış, temizlenmiş veya ağartılmış	ipek, beyaz, yık	1
5	değişik renkli dokuma kumaşlar iplikler	[GTIP NOsu YOKTUR]: Farklı renkteki ipliklerden (5007 altında)	kumaş, renk, iplik, değişik	2
6	ipek ipliği ile dokunmuş kumaşlar	5007: İpek veya ipek döküntülerinden dokunmuş mensucat	ipek, dokun, kumaş	2
7	el tezgahlarında dokunmuş ipek kumaşlar	5007.10.00.00.11: El tezgahlarında dokunanlar	dokun, kumaş, ipek, dokun, el, tezgah	1
8	el tezgahlarında dokunmuş duvar halısı	58: Özel dokunmuş mensucat; tuftedilmiş dokumaya elverişli mensucat; dantela, duvar halıları; şeritçi ve kaytancı eşyası; işlemler	halı, dokun, duvar	2
9	kırpılmış hayvan kılından dokuma eşya	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	kırp, kıl, hayvan	1
10	suni malzemedan yapılmış iplikler lifler	55: Sentetik ve suni devamsız lifler	suni, lifle	1
11	suni dokuma için kullanılan ipler	54: Sentetik ve suni filamentler, şeritler ve benzeri sentetik ve suni dokumaya elverişli maddeler	dokuma, suni	3

5 keywords Query Results (Continued)				
No	Query Sentence	Target Product	Matched Keywords	Rank
12	dokumaya elverişli suni akrilik maddeler	54: Sentetik ve suni filamentler, şeritler ve benzeri sentetik ve suni dokumaya elverişli maddeler	dokuma, suni, elveriş	2
13	tığ ile örülmüş dentel aksesuar	61: Örme giyim eşyası ve aksesuarı	tığ, aksesuar	1
14	şiş ile örülmüş gevşek kumaş	60: Örme Mensucat	ör, şiş	1
15	yapağı ve yün iplikten atkı	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	iplik, yün, yapağı	1
16	hayvan postundan kırılmış yünlü kıyafet	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	kırp, hayvan, yün	1
17	bitkisel lif ve ipliklerden yapılma giysi	53: Dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat	bitkisel, iplik, lif	1
18	halı ve yere serilen eşyalar	63: Dokumaya elverişli maddelerden diğer hazır eşya; takımlar; kullanılmış giyim eşyası ve dokumaya elverişli maddelerden kullanılmış eşya; paçavralar	halı, yer	1
19	kullanılmamış dokunmuş giyim giysi kıyafet	62: Örülmemiş giyim eşyası ve aksesuarı	giyim, kıyafet, giysi	1
20	saf ipekten kumaşlar bezler dokumalar	[GTIP NOSu YOKTUR]: Saf ipekten ponje, habutai, honan, şantuk, korah ve benzeri diğer Uzak Doğu kumaşları (buret ve diğer ipek döküntüleri veya dokumaya elverişli maddeler karıştırılmamış)	kumaş, ipek, dokuma, saf, kumaş	2
21	tığ ile yapılmış aksesuar süs	61: Örme giyim eşyası ve aksesuarı	aksesuar, tığ	1
22	at kılından dayanıklı ip iplik	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	iplik, kıl, at	1
23	tiftik keçisi yününden iplik ip	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	tiftik, keçi, yün, iplik, ip	2

<i>5 keywords Query Results (Continued)</i>				
No	Query Sentence	Target Product	Matched Keywords	Rank
24	kumaş ip parlak ince pahalı	5007.10: Burekten mensucat	kumaş, parlak, ince	17
25	ipekten dokunmuş ince kumaş dokuma	5007.10.00.00.11: El tezgahlarında dokunanlar	dokun, kumaş, ince, ipek	30

APPENDIX F

<i>3 keywords with 1 negativity Query Results</i>				
No	Query Sentence	Target Product	Matched Keyword	Rank
1	ipek olmayan iplik	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	iplik	3
2	ipek olmayan kıyafet	61: Örme giyim eşyası ve aksesuarı	kıyafet	3
3	döküntü olmayan ipek	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	ipek	3
4	döküntü hariç iplik	53: Dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat	iplik	3
5	hayvansal olmayan iplik	53: Dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat	iplik	2
6	döküntüsüz ipek	5002.00.00.00.00: Ham ipek (bükülmemiş)	ipek, ham	2
7	elverişli olmayan ip	56: Votka, keçe ve dokunmamış mensucat; özel iplikler; sicim, kordon, ip, halat ve bunlardan mamul eşya	İp	1
8	iplik olmayan ipek	5007.20.41.00.00: Seyrek dokunmuş	ipek	7
9	iplik olmayan ipek	5002.00.00.00.00: Ham ipek (bükülmemiş)	ipek	14
10	iplik hariç ipek	[GTIP NOsu YOKTUR]: Saf ipekten ponje, habutai, honan, şantuk, korah ve benzeri diğer Uzak Doğu kumaşları (buret ve diğer ipek döküntüleri veya dokumaya elverişli maddeler karıştırılmamış)	ipek	15
11	parekande hariç ipek	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	ipek	18
12	sentetik olmayan lif	53: Dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat	lif	1

<i>3 keywords with 1 negativity Query Results (Continued)</i>				
No	Query Sentence	Target Product	Matched Keyword	Rank
13	lif olmayan sentetik	54: Sentetik ve suni filamentler, şeritler ve benzeri sentetik ve suni dokumaya elverişli maddeler	sentetik	1
14	aksesuar olmayan örgü	60: Örme Mensucat	örgü	1
15	kullanılmış olmayan kıyafet	61: Örme giyim eşyası ve aksesu	kıyafet	2
16	bitkisel olmayan iplik	51: Yapağı ve yün, ince veya kaba hayvan kılı; at kılından iplik ve dokunmuş mensucat	iplik	9
17	mensucat olmayan ipek	5004.00 : İpek ipliği (ipek döküntülerinden elde edilen iplikler hariç) (perakende satılacak hale getirilmemiş)	iplik	7
18	mensucat olmayan ipek	5005.00: İpek döküntülerinden elde edilen iplikler (perakende satılacak hale getirilmemiş)	ipek	9
19	tüy olmayan iplik	53: Dokumaya elverişli diğer bitkisel lifler; kağıt ipliği ve kağıt ipliğinden dokunmuş mensucat	iplik	3
20	tüy olmayan iplik	5005.00: İpek döküntülerinden elde edilen iplikler (perakende satılacak hale getirilmemiş)	iplik	8
21	koza olmayan ipek	5002.00.00.00.00: Ham ipek (bükülmemiş)	ipek	15
22	iplik olmayan koza	5001.00.00.00: Çekilmeye elverişli ipek böceği kozaları	koza	4
23	ipek olmayan koza	52: Pamuk	koza	1
24	ipek olmayan dokuma	57: Halılar ve dokumaya elverişli maddelerden diğer yer kaplamaları	dokuma	2
25	ipek olmayan mensucat	60: Örme Mensucat	mensucat	3

APPENDIX G

<i>Keywords extracted from the sample domain data</i>				
Letter	Noun keywords	Verb keywords	Adjective keywords	Keywords created by the designer
<i>A</i>	ađır	ađar		Akrilik
	aksesuar	ađartılma		
	at			
	ayak			
<i>B</i>	baskı	bük		beyaz
	benzer	bükölme		bitkisel
	bez			Bursa
	bitkisel			
	boya			
	böcek			
<i>C</i>	cm			
<i>Ç</i>		çekilme		çöp
<i>D</i>	daha	ditme	düz	dayanık
	dahil	dokun		dođal
	dantel			
	dantela			
	devam			
	diđer			
	dođu			
	dokuma			
	döküntü			
	duvar			
	<i>E</i>	el	emdir	
elveriř				elbise
				esnek
<i>F</i>	fark		fazla	
<i>G</i>	geçen	geçme		giysi
	geniřlik	getirilme		
	giyim	görme		
	gudde			
<i>H</i>	halat			hafif
	halı			
	hayvan			
	hazır			
	honan			
<i>İ</i>	ileri	içer	ince	iğne
	iřlem	iřleme		ilme

<i>Keywords extracted from the sample domain data (continued)</i>				
Letter	Noun keywords	Verb keywords	Adjective keywords	Keywords created by the designer
	ip			
	ipek			
	iplik			
	işlem			
<i>K</i>	kağıt	kaplama	kaba	kırp
<i>K</i>	kaytan	Kaplan		kırpıntı
	keçe	karıştırılma		kıyafet
	kıl	kullan		kimyasal
	kordon			
	koza			
	krep			
	kumaş			
<i>L</i>	lamine	Lifle		lateks
	lif			
<i>M</i>	mamul			
	mensucat			
	misina			
<i>N</i>				naylon
<i>Ö</i>	özel	ör		
		örülme		
<i>P</i>	paçavra			parlak
	pamuk			paçavra
	perakende			polyester
	parekende			
	ponje			
<i>R</i>	renk			
<i>S</i>	santimetre		saf	sünger
	sentetik		seyrek	
	sıva			
	sicim			
	sunı			
<i>Ş</i>	şerit	Şeritle		şiş
<i>T</i>	takım	temizle	temiz	tığ
	teknik	temizleme		tüy
	tekstil	Tiftme		
	tezgah			
	tufte			
<i>U</i>			uzak	
<i>V</i>	vatka			
<i>Y</i>	yapağı			yumuşak
	yer			
	yün			

Special Keywords for NOT Querying: hariç, olmayan, -siz suffix, -me/ma suffix

