UNIVERSIDADE DE ÉVORA

ESCOLA DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

Ontology-Based Information Extraction from Learning Management Systems

Rodwan Bakkar Deyab

Orientação *Prof. Irene Rodrigues*

Mestrado em Engenharia Informática

Dissertação

Évora, 2017

UNIVERSIDADE DE ÉVORA

ESCOLA DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE INFORMÁTICA

Ontology-Based Information Extraction from Learning Management Systems

Rodwan Bakkar Deyab

Orientação *Prof. Irene Rodrigues*

Mestrado em Engenharia Informática

Dissertação

Évora, 2017

For my dear mother

Acknowledgements

All sincere thanks to my professor Irene Rodrigues who was very patient and helpful to me. All sincere thanks to Helena Barroco who made it possible for me to come to live and study in this beautiful university. All sincere thanks to President Jorge Sampaio the founder of the Global Platform for Syrian Students which provided and still providing all kind of support.

Contents

List of Figures

List of Tables

Acronyms

ECT Escola de Ciências e Tecnologia

UE Universidade de Évora

LMS Learning Management System

SW Semantic Web

RDF Resource Description Framework

OWL Web Ontology Language

IE Information Extraction

OBIE Ontology-Based Information Extraction

OP Ontology Population

OL Ontology Learning

GATE General Architecture for Text Engineering

JAPE Java Annotation Patterns Engine

NLP Natural Language Processing

POS-Tagging Part-of-Speech Tagging

NER Named Entity Recognition

QA Question Answering

SPARQL Simple Protocol and RDF Query Language

Abstract

In this work we present a system for information extraction from Learning Management Systems. This system is ontology-based. It retrieves information according to the structure of the ontology to populate the ontology. We graphically present statistics about the ontology data. These statistics present latent knowledge which is difficult to see in the traditional Learning Management System. To answer questions about the ontology, a question answering system was developed using Natural Language Processing in the conversion of the natural language question into an ontology query language.

Keywords: Semantic Web, Learning Management Systems, Information Extraction, Ontology Population, Natural Language Processing, Question Answering Systems

Sumário

Extração de Informação de Sistemas de Gestão para Educação Usando Ontologias

Neste dissertação apresentamos um sistema de extracção de informação de sistemas de gestão para educação (Learning Management Systems). Este sistema é baseado em ontologias e extrai informação de acordo com a estrutura da ontologia para a popular. Também permite apresentar graficamente algumas estatísticas sobre os dados da ontologia. Estas estatísticas revelam o conhecimento latente que é difícil de ver num sistema tradicional de gestão para a educação. Para poder responder a perguntas sobre os dados da ontologia, um sistema de resposta automática a perguntas em língua natural foi desenvolvido usando Processamento de Língua Natural para converter as perguntas para linguagem de interrogação de ontologias.

Palavras chave: Web Semântica, Sistemas de Gestão para a Educação, Extração de Informação, População de Ontologias, Processamento de Língua Natural, Sistemas de Resposta Automática a Perguntas

1

Introduction

Learning Management Systems are applications which provide services like administration, reporting and documentation for online courses. The traditional Learning Management Systems which depend on relational databases have limits in the information they provide to the user. Improving the Learning Management Systems can be achieved by building other structures to store their data. Structures like ontologies can provide better performance and more flexible way to access information. To achieve that, Information Extraction can help in extracting the data from databases to be populated to ontologies. Having the ontology, we can exploit it making question answering systems which can answer questions in natural language question using Natural Language Processing to achieve that.

1.1 Objectives

The objectives of this work are: to build and populate an ontology with information extracted from the Moodle Learning Management System, the data of the e-learning courses of the University of Évora. Then to build a question answering system to answer natural language questions about the ontology data showing that these question were not possible to answer in the traditional Learning Management System. The system should allow us to retrieve information on the courses and student performance.

1.2 Principal Contributions of This Dissertation

In this dissertation we developed an information extraction system that retrieves information from Learning Management Systems(LMS) into an ontology. We present, graphically, some statistics about the ontology data after populating it with the extracted information. We developed a question answering system to query the ontology in natural language. It uses Natural Language Processing to convert the question to SPARQL.

1.3 Structure of The Dissertation

This dissertation is composed of five chapters:

Chapter 2: a state of the art about Semantic Web, Natural Language Processing, Information Extraction and Ontology Population.

Chapter 3: a system of extracting educational information from the Moodle Learning Management System to populate an ontology.

Chapter 4: graphical statistics about the content of the populated ontology. These graphs gives a general view and an easy way to understand the content of the ontology and make some conclusions.

Chapter 5, a question answering system which answers questions about the ontology content. This system uses Natural Language Processing to convert the questions to SPARQL query to run on the ontology and shows the answer graphically.

Chapter 6: a conclusion and future work.

2

State of the Art

In this state of the art we present the Semantic Web and the Web Ontology Language (OWL). This language is used to build ontologies. We also present Ontology Population using Information Extraction and Natural Language Processing.

2.1 Semantic Web

Semantic Web was proposed by Sir Tim Berners-Lee, the creator of the World Wide Web and the director of the W3C (World Wide Web Consortium). So it can be best defined using his own words:

> A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities [BLHL+01]

Consider the text: "Obama is the president of the US". Figure 2.1, presents graphically an interpretation of the text using OWL in Protegé (to be presented in next sections).

- 1 and 2: are subclass relations.

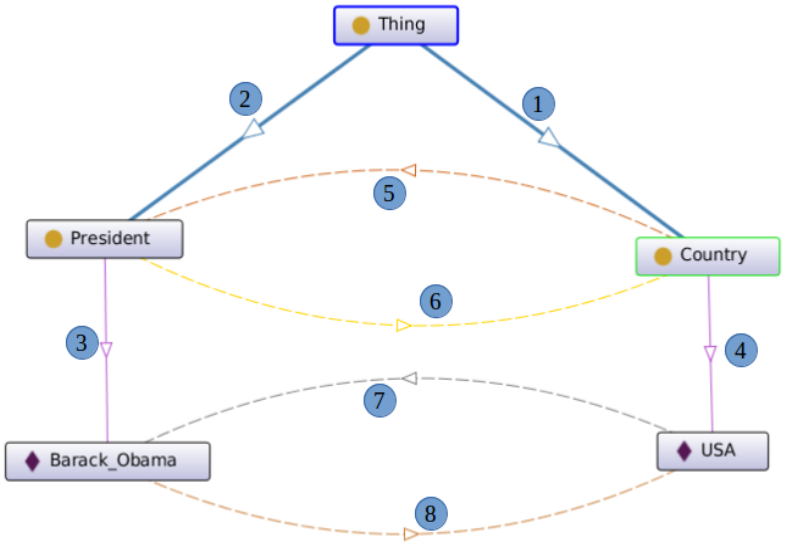

Figure 2.1: A graphical semantic representation for the sentence

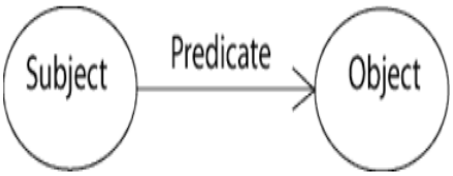

Figure 2.2: RDF data model

- 3 and 4: are instance relations.
- 5,7 and 6,8: “hasPresident”, “isPresidentOf” relations (respectively).

So the text was converted into a structure with classes (Thing, President and Country), individuals (Barack_Obama and USA) and relations to build a meaningful structure.

2.1.1 Web Ontology Language (OWL), RDF & RDFS

OWL is built upon RDF. RDF (Resource Description Framework)[KC06], is a model for representing information. This model defines a collection of connected triples. Each triple is (subject, predicate, object). Figure 2.2 presents this model. Each triple defines a relationship between a subject and an object. The subject like “Barack Obama”, the predicate (attribute) like “isPresidentOf” and the Object like “USA”. This model has some limitations. For example, it is not capable of defining two classes to be disjoint, like Female and Male classes.

```
<owl:Class rdf:about="NS#Course"/>
```

Figure 2.3: Define a class in OWL

```
owl:  http://www.w3.org/2002/07/owl#
rdf:  http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
```

Figure 2.4: Standard namespaces for OWL

RDFS (RDF Schema)[1] *"is a general-purpose language for representing simple RDF vocabularies on the Web".* For example when defining a new class to be a subclass of another one, the subclass definition comes from the RDFS namespace. And by defining this under the RDFS namespace it will be uniquely recognized.

OWL [AVH04] is a knowledge representation language for ontologies. Ontologies will be presented in the next section. OWL has more expressive power than RDF. OWL has three types:

1. **OWL Full:** It is the more expressive type of OWL. It is fully compatible with RDF and RDF Schema. Sometimes it is undecidable and difficult to apply reasoning on it. Reasoning in owl is the process of extracting facts that are not present in the ontology explicitly.
2. **OWL DL:** OWL Description Logic. It is a sublanguage of OWL Full. It allows better reasoning but it is not fully compatible with RDF. An RDF document needs to be modified to be a legal OWL DL document but every OWL DL document is a legal RDF document.
3. **OWL Lite:** It is more restricted than OWL DL. It has less expressive power but it is easier to understand and to implement.

OWL Elements

1. **Class:** The class element is used to define an abstract concept. A class can refer to a person, location, etc.

   Figure 2.3 presents an OWL definition for "Course" class. NS is the name space of the ontology. The name space will make the ontology unique and recognized among other ontologies.

   Some standard namespaces are "owl", "rdf" and "rdfs". They are presented in figure 2.4.

   OWL can represent a hierarchy of classes. For example, a class "Man" is a subclass of the class "Person". This is presented in figure 2.5

2. **Property:** OWL has two type of properties:

[1] http://www.w3.org/2001/sw/wiki/RDFS

```
<rdf:Description rdf:about="NS#Man">
    <rdf:type rdf:resource="owl#Class"/>
    <rdfs:subClassOf rdf:resource="NS#Person"/>
</rdf:Description>
```

Figure 2.5: Defining class hierarchy in OWL

```
<rdf:Description rdf:about="NS#inUniversity">
  <rdfs:range rdf:resource="NS#University"/>
  <rdfs:domain rdf:resource="NS#Course"/>
  <rdf:type rdf:resource="owl#ObjectProperty"/>
</rdf:Description>
```

Figure 2.6: Object property in OWL

```
<rdf:Description rdf:about="NS#Economics_And_Finance">
    <rdf:type rdf:resource="NS#Course"/>
    <inUniversity rdf:resource="NS#University_of_Michigan"/>
</rdf:Description>
```

Figure 2.7: Object property in OWL relates two individuals

(a) Object Properties relate classes to each other. Considering the two classes "Course" and "University", to represent that the course is taught in a university, we use the object property "inUniversity". Figure 2.6 presents this object property in OWL. Each object property has a domain class and a range class. "Course" is the domain class of the object property "inUniversity" and "University" is the range.

Figure 2.7 presents two individuals "Economics_And_Finance" and "University_of_Michigan" with the classes "Course" and "University" respectively. The are related by the object property "inUniversity".

(b) Data type properties relate objects to literals. They allow to represent relations between classes and literals. For example, figure 2.8 presents a data type property "hasID" for the individual "Chemistry" of the class "Course".

3. **Individual:** Individuals are the instances of the abstract concepts represented by classes in the ontology. Figure 2.8 presents an individual "Chemistry" of the class "Course".

In OWL it is possible to [HKR+04]:

- Define classes to be equivalent or to be disjoint.
- Make restrictions on object properties to range in specific domains.
- Make restrictions on data property to have specific values.
- Restrict object properties to be "functional", "inverseOf", "transitive" or "symmetric".
- Define properties to be sub properties from others.

```
<rdf:Description rdf:about="NSChemistry">
    <rdf:type rdf:resource="NSCourse"/>
    <hasID>1678</hasID>
</rdf:Description>
```

Figure 2.8: Data type property in OWL

2.1.2 Ontology

In philosophy, *"ontology is the part which is concerned in understanding the nature of existence"*.

In computer science, ontology is defined as[Gru93]:

"A specification of a representational vocabulary for a shared domain of discourse — definitions of classes, relations, functions, and other objects — is called an ontology"

And formally [SM01]:

A (core) ontology is a tuple $\Omega := (C, is\_a, R, \sigma)$ where C is a set whose elements are called *concepts*, is_a is a partial order on C (i.e., a binary relation $is\_a \subseteq C \times C$ which is reflexive, transitive, and anti-symmetric), R is a set whose elements are called *relation names* (or *relations* for short), and $\sigma : R \to C^+$ is a function which assigns to each relation name its arity.

Some of the ontologies advantages are[2]:

- To share common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge:
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.
- To analyse domain knowledge.

Some available ontologies are:

- DBPedia[3]:

  "It is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link the different data sets on the Web to Wikipedia data.....Altogether the DBpedia 2014 release consists of 3 billion pieces of information (RDF triples) out of which 580 million were extracted from the English edition of Wikipedia".

- Gene Ontology[4]:

  "It is a collaborative effort to address the need for consistent descriptions of gene products across databases".

- WordNet[5]:

  "It is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations".

In this work an ontology for Moodle Learning Management System was built.

[2] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
[3] http://wiki.dbpedia.org/about
[4] http://geneontology.org/page/documentation
[5] https://wordnet.princeton.edu/

```
S1  P1  O1.
S2  P2  O2.
....
....

SELECT ?var WHERE {S1  P1 ?var}
```

Figure 2.9: A SPARQL query and RDF file

2.1.3 Simple Protocol and RDF Query Language (SPARQL)

SPARQL is an RDF query language. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium[6]. As RDF represents knowledge by triples (subject, predicate and object), SPARQL depends in its search methodology on matching these triples. Figure 2.9 presents an RDF file and a SPARQL query. By running this SPARQL query on this file, the query will match all the triples which have S1 and P1 as subject and predicate and return the value of the object. It can retrieve zero or more matches.

SPARQL has features like:

- It allows matching literals with language tags (matching literals in English language for example).
- It allows matching literals with numerical types and arbitrary data types.
- It allows restricting the values of strings like applying filters (regular expressions for example).
- It allows restricting the numeric values like applying filters which can restrict on arithmetic expressions.
- It allows optional values.
- It can query more than one ontology at the same time. This gives the ability to retrieve information from different sources.

Jena API[7] is *"a free and open source Java framework for building Semantic Web and Linked Data applications"*. It is used in this work to run SPARQL queries to retrieve information from the ontology.

Some ontologies like DBPedia have "SPARQL Endpoints" which enables us to run SPARQL queries online.

2.1.4 Protégé

Protégé[8] *"is a free, open-source ontology editor and framework for building intelligent systems".* It was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. It fully supports the latest OWL 2 Web Ontology Language and RDF specifications from the World Wide Web Consortium. It provides a graphical interface to build and edit ontologies. It enables reasoning on the ontology to make sure of its consistency. It accepts many plugins to provide more functionalities like OntoGraf[9] which provides a graphical representation for the ontology.

Protégé is used in this work to build and edit the ontology. OntoGraf is used to present the ontology graphically.

[6] https://www.w3.org/
[7] https://jena.apache.org/
[8] http://protege.stanford.edu/
[9] http://protegewiki.stanford.edu/wiki/OntoGraf

2.2 Natural Language Processing(NLP)

Natural Language Processing is an area of research considered as a subfield of Artificial Intelligence. This area is concerned about making the computer understand natural language text and its meaning. It is an interdisciplinary such that it depends on many disciplines like machine learning, linguistics, mathematics, artificial intelligence, information science and psychology. In order to achieve its tasks, understanding of phonetics, morphology, grammar, lexicology and semantics should be present. Some reasons why NLP is difficult are:

- Ambiguity: consider the sentence :"I saw my friend in the classroom with a laptop." This sentence can be interpreted in many different ways:
  - I saw my friend. He was in the classroom. My laptop was with me.
  - I saw my friend. I was in the classroom. My laptop was with me.
  - I saw my friend. He was in the classroom. His laptop was with him.
  - I saw my friend. We were in the classroom. My laptop was with me.
  - I saw my friend. We were in the classroom. His laptop was with him.
- Natural language is related to the psychological state of the human. We can say exactly the same word in different ways to mean different things.

NLP has many tasks, tools and applications. This will be presented in the next sections.

2.2.1 Major Tasks in NLP

NLP has tasks like Part-of-Speech(POS)-tagging, Named Entity Recognition (NER) and Syntactic Analysis (Parsing). These tasks can be achieved using many approaches, for example, rule-based and machine learning approaches.

Part-of-Speech(POS) Tagging

It is assigning the syntactical part of speech for each word in a sentence. For example, it tags a word as a verb, noun, adjective, etc. This task is useful for other tasks like Named Entity Extraction(NER) which will be explained in the next section. POS Tagging can be achieved by many approaches[KJ15]:

- Supervised Taggers:
  - Rule-Based: Brill Tagger
  - Stochastic: Hidden Markov Model(HMM)
  - Neural Network
- Unsupervised Taggers:
  - Rule-Based: Brill Tagger
  - Transformation-Based: User Baum-welch
  - Neural Network

In this work, Stanford POS-tagger[TM00, TKMS03] was used. This will be presented in the next sections.

Figure 2.10: Location detection by looking up a gazetteer

Named Entity Recognition NER

Named entity refers to an entity like place, organization, person, date, etc. NER works on defining the named entities in the textual context. It defines the boundaries on the named entity (some entities can be more than one word like “New York”, for example).

A basic approach for recognizing these named entities is to use a gazetteer which contains a comprehensive list of the possible entities in a specific domain. This approach may give wrong results. For example [BKL09], figure 2.10 presents the result of identifying all the locations in a text. It shows the gerund “Reading” as a city in the UK. Actually, Reading is a town in the county of Berkshire in England. As the gazetteer is rich enough to contain the name of this city, it is naive enough to take that decision. It does not matter how rich the gazetteer is, it will not be comprehensive like in case of names of people and in case of organizations, we always have new names appearing.

Machine Learning is used to achieve NER. There are three approaches: supervised learning, semi-supervised learning and unsupervised learning:

- Supervised Learning:

  This method needs an annotated corpus to learn the rules of detecting entities. The supervised learning method includes: Hidden Markov Models (HMM)[ZS02], Decision Trees [SFK06], Maximum Entropy Models [CN02], Support Vector Machines (SVM) [Mic13], and Conditional Random Fields (CRF) [Set04].

- Unsupervised Learning:

  This method depends on clustering and does not need any training data. Clustering is the process of grouping all the similar objects in a group called a cluster. Some examples are: E. Alfonseca and Manandhar study [AM02] and Y. Shinyama and Sekine [SS04]

- Semi-supervised Learning:

  Semi-supervised learning is halfway between supervised and unsupervised learning. In this method, the system has some training examples but it does not cover all the possibilities or even a small portion of

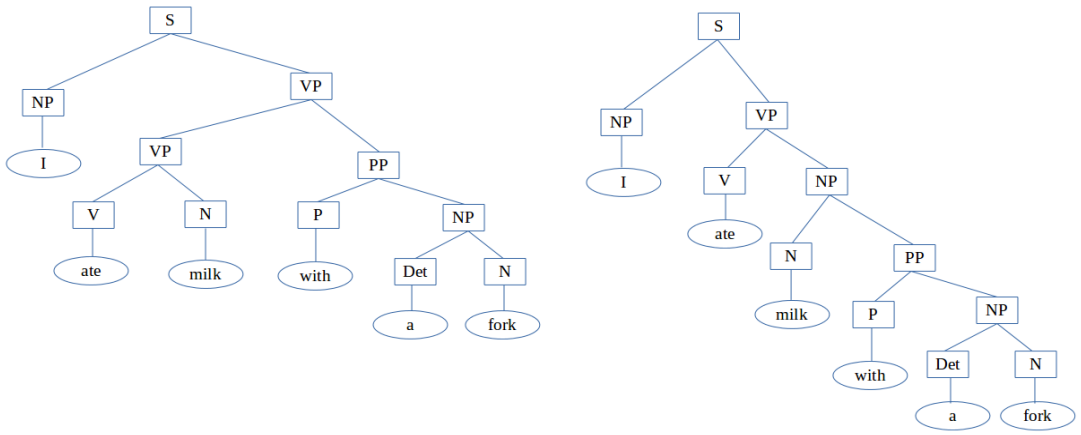

Figure 2.11: Two parsing trees for an ambiguous sentence

them. Some of the approaches which uses this method are: S. Brin [Bri99] and E. Riloff and Jones [RJ+99]

Syntactic Analysis(Parsing)

Syntactic Analysis determines the parsing tree of the sentence. As natural language is ambiguous, some sentences can have more than one parsing tree. Figure 2.11 shows two possible parsing trees for this sentence "I ate milk with a fork".

An approach to achieve the syntactic analysis is the Probabilistic Context Free Grammar(PCFG) [KM03].

Dependency Parsing

Dependency parsing determines the syntactic relations between words in the sentence. It resolves ambiguity. For example, an ambiguous sentence "I saw my friend with glasses". This sentence has two possible meanings:

- I saw my friend, I was using glasses.
- I saw my friend, he was using glasses.

Using the Stanford Dependency Parser Online Demo[10], figure 2.12 presents the dependency parsing of the sentence. This dependency parsing shows that the sentence has the second meaning removing the ambiguity.

Some dependency parsing approaches are:

- Shift-reduce [ST08].
- Spanning tree [MPRH05].

[10] http://nlp.stanford.edu:8080/corenlp/

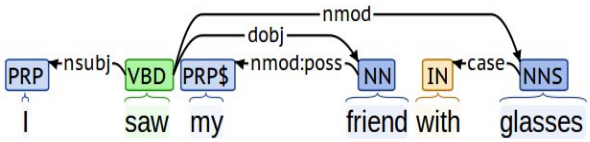

Figure 2.12: Dependency parsing example

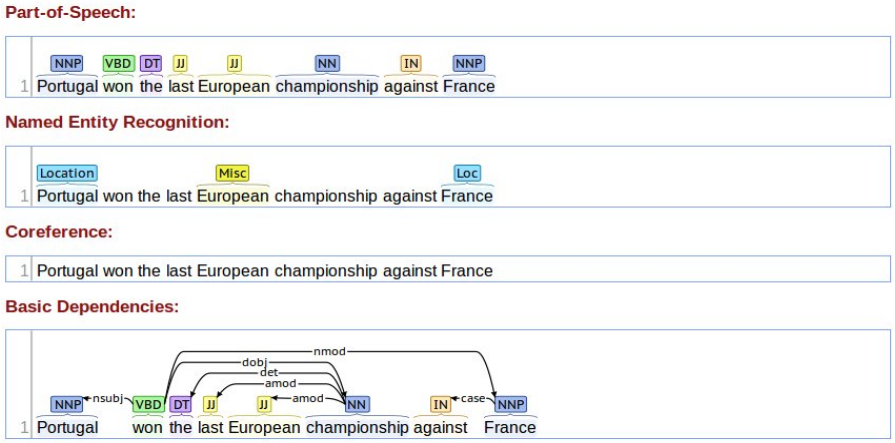

Figure 2.13: Stanford CoreNLP Online Demo example

- Cascaded chunking [KM02].

In this work, the Stanford Dependency Parser [CM14] was used.

2.2.2 NLP Tools and Applications

There are many tools for NLP, Stanford NLP tools[11], Apche OpenNLP[12] and NLTK[13]. In this work, Stanford NLP was used with the GATE framework which will be presented in the next sections. Stanford NLP[14] *"is a set of natural language analysis tools which can take raw text input and give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, etc".*

Figure 2.13 presents an example of the Stanford CoreNLP online demo[15]. The sentence used as an example: "Portugal won the last European championship against France". Some of the Stanford NLP tools are:

- Stanford POS-Tagger: is an implementation of the log-linear part-of-speech taggers described in [TM00, TKMS03].
- Stanford NER: is an implementation of linear chain Conditional Random Field (CRF) sequence models [FGM05].

[11] http://nlp.stanford.edu/
[12] https://opennlp.apache.org/
[13] http://www.nltk.org/
[14] http://nlp.stanford.edu/
[15] http://nlp.stanford.edu:8080/corenlp/

```
team1:      Juventus
country1:   Italy
team2:      Barcelona
country2:   Spain
year:       2015
result:     3-1 (Barcelona-Juventus)
```

Figure 2.14: Information extraction result example

- Stanford Dependency Parser: presented in [CM14], uses Neural Network approach.

Some NLP applications are:

- Spell checking: like chat mobile applications which correct the spelling automatically.
- Question answering systems: these systems works on extracting understandable machine commands from a natural language question.
- Machine Translation: like google translate.
- Sentimental Analysis: like determining the opinion of the writer.
- Speech Recognition: is the translation of spoken language into textual one.
- Intelligent Web Searching.

2.3 Information Extraction(IE)

Information extraction is the process of extracting structured information from unstructured, semi-structured or structured information. Information extraction systems are domain-dependent such that to achieve good testing results they should be trained and tested on corpora of the same domain.

Information extraction system produces a structured result of the extracted information. It fills information in a predefined structure. Consider a text taken from sport report about the final in the European champions league: "Juventus from Italy and Barcelona from Spain played the final of the European champions league for 2015 and the result was 3-1 for the Spanish team". Figure2.14 presents the result of information extraction from this text depending on an information extraction system in the domain of "sport reports about the European champions league".

2.3.1 IE Sources, Structures, Methods and Tools

Information Extraction Systems can extract information from data sources like:

- Structured: like databases.
- Semi-structured: like xml files.
- Unstructured: like texts.

They can extract:

- Entities: like person, organization, location, etc.
- Relationships: between the entities extracted like an organization which is located in a place.

Methods of Extraction: [Sar08]

- Hand-coded or Learning-based: *"A hand-coded system requires human experts to define rules or regular expressions or program snippets for performing the extraction. That person needs to be a domain expert and a programmer, and possess descent linguistic understanding to be able to develop robust extraction rules. In contrast, learning-based systems require manually labelled unstructured examples to train machine learning models of extraction".*
- Rule-based or Statistical: *"Rule-based extraction methods are driven by hard predicates, whereas statistical methods make decisions based on a weighted sum of predicate firings. Rule-based methods are easier to interpret and develop, whereas statistical methods are more robust to noise in the unstructured data".*

IE systems still face many challenges like accuracy such that they are not highly accurate, high running time and being dependant on specific domains.

Some Information Extraction tools:

- GATE [Cun02]
- NLTK
- OpenNLP
- Stanford NLP

NLTK, OpenNLP and Stanford NLP are NLP tools but they can be adapted to achieve Information Extraction. They all do NER which is considered also as an IE task. GATE is an Information Extraction framework but it can be used as an NLP framework.

In this work, the system extract information from a structured data source which is the database of the Moodle Learning Management System. This will be presented in the next chapters. GATE [Cun02] framework was used to achieve the information extraction.

2.3.2 IE Applications

Information Extraction systems are useful in many areas [Sar08]:

- Enterprise applications like:
  - Customer care, where it generates many structured forms from customer interactions. These forms are used for achieving better management.
  - Data cleaning, like in the process of data warehouse cleaning. It is necessary to convert the addresses which are represented using plain texts into structured forms. This is useful for organizations which have millions of addresses like banks and telephone companies.

- Personal Information Management: organizing personal data in structured format.
- Scientific Applications: bio-informatics.
- Web Oriented Applications: like the comparison between shopping websites.

2.3.3 GATE IE System

GATE [Cun02] is an information extraction framework. GATE is a Hand-coded IE system which depends on rules written in JAPE[16]. It can include NLP tools as plugins to benefit them in the extraction process. In this work, Stanford NLP tools were included in the GATE framework to achieve NLP. GATE was used in this work to process natural language questions to convert them to SPARQL queries. This will be explained in the chapter 5.

Information extraction can be based on an ontology. This means that the extraction process will extract entities to fill inside the predefined ontology structure such that the ontology guides the extraction process. This will be presented in the next section.

2.4 Ontology Information Extraction Systems from Text

A general structure of an ontology information extraction system from text is proposed. The following terms denote, roughly, the same concept in the literature: Ontology Population (OP), Ontology-Based Information Extraction (OBIE) and Ontology-Driven Information Extraction (ODIE). State-of-the-art systems with different techniques to achieve the ontology population from text are presented. A comparison between these systems according to some proposed criteria is discussed.

2.4.1 Ontology Population System Architecture

In figure 2.15 we present a general structure of a *System of Ontology Population from Text*. The main modules of this system are:

Corpus The data set from which the information will be extracted. The web is a good source to build corpora for a specific domain by crawling and collecting information. Many corpora were built by crawling the web in many domains like tourism *Ruiz-Martínez et al.* [RMMGCN+11] and sport [BCRS06].

Input Ontology The empty input ontology which will be populated by the information extracted. The ontology could be built manually or automatically. The process of obtaining the ontology from a specific domain is called *Ontology Learning* [WLB12]. The ontology leads the extraction process such that the information extracted should correspond to the ontology terms.

Preprocessing & Extraction Module Preprocessing the input corpus facilitates the extraction task. Tokenizing and stemming are common preprocessing tasks. Information Extraction is presented in section 2.3.

Population Module The information extracted will be classified and populated the corresponding concepts and relations represented in the ontology. Entity Disambiguation and Synonymy Resolution are used in this process.

[16] https://gate.ac.uk/sale/tao/splitch8.html

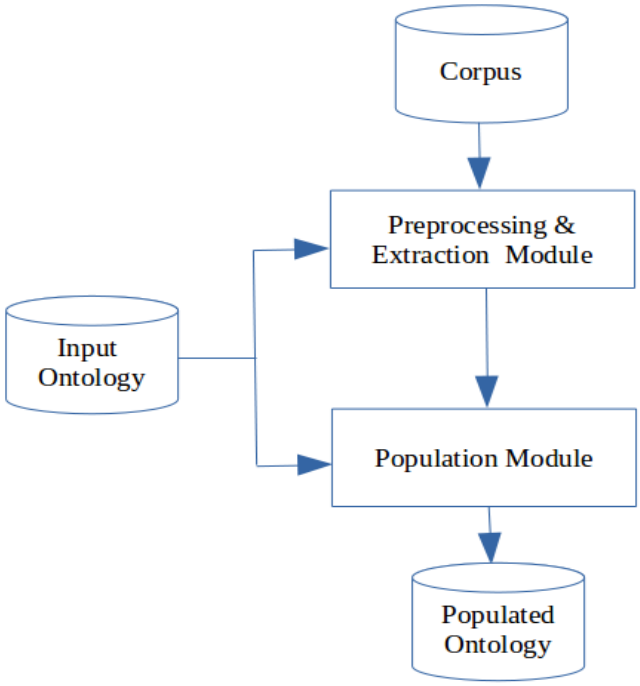

Figure 2.15: A General Architecture for a *System of Ontology Population from Text*

Populated Ontology The populated ontology with the information extracted.

In the next section, we present some Ontology Population Systems.

2.4.2 Ontology Population Systems

The presented systems populate ontologies with information extracted from text. Some of these system build the ontology automatically and some of them have manually built ontologies.

SOBA [BCRS06] The SmartWeb Ontology-Based Annotation (SOBA) system is a sub-component of the SmartWeb system. This system was developed to populate the SWInto (SmartWeb Integrated Ontology) ontology [OAH+07]. This system can extract instances and relations between them from free text (sport reports and image captions) and tables (semi-structured) from the web. It was applied for the sport domain from the FIFA and UEFA websites covering matches of the world cup 2002 and 2006. This system uses Part-Of-Speech Tagging and NER techniques to build grammar to achieve the extraction process. It exploits the SProUT system [DKP+04] to build the extraction grammar. SProUT is able to extract entities like persons, locations, numerals and date and time expressions. The rule set of SProUT was extended with gazetteers, part-of-speech and morphological information. More grammar were developed to extract entities related to the sport domain like players, referees, etc. SOBA uses the OntoBroker system [DEFS99] as a reasoner.

Text-To-Onto [MS00] This system was developed to learn and populate its own ontology for the German language in a specific domain. It uses many techniques like POS tagging, NER, chunk parsing and machine learning.

Faria and Girardi [FG11] This system was applied on a corpus of the Family Law domain [17]. It uses linguistic

[17] http://family.findlaw.com/

rules to extract instances and relations to populate the LawFamily ontology which was built manually using Protégé [NFM00]. It exploits GATE [Cun02] and JAPE rules to achieve the task. Mainly, it does Part-Of-Speech Tagging and Named Entity Recognition.

Ruiz-Martínez et al. [RMMGCN$^+$11] This system is applied in the eTourism domain for Spanish language in Spain. It uses GATE [Cun02] and JAPE grammar to achieve the task. It mainly uses Named Entity Recognition. This system has a reasoning phase during and after the process of *Ontology Population* to maintain the consistency of the information integrated into the ontology. It uses the HermiT reasoner[18]. This system was developed for Spanish Language but it can be adaptable to other languages and other domains. The ontology used for this system is *travel.owl* which is manually built using Protégé [NFM00]. This ontology was enriched with some more classes about Spanish Hotel Industry from the *OnTour ontology* [Pra04].

Ontopop [TL12] This system provides a plug-in for Protégé [NFM00]. It takes as an input an ontology which was developed by the GETESS project [SBB$^+$99]. It is in the tourism domain and it has 682 concepts with no instances. It was populated with some training instances from the YAGO ontology [SKW08]. The ontology population in this system depends on the training instances to retrieve sentences from the wikipedia database. These retrieved sentences are used to extract some syntactic features for the instances. The syntactic features are extracted using dependency parsing by exploiting the Stanford Parser[19]. Named Entity Recognition is then applied. Then it exploits the Latent Semantic Analysis to detect polysemy and synonymy. Its performance was tested on 30 corpora collected from the internet[20].

Sadoun et al. [SDGDG13] This system was developed for the *smart space* domain. The corpus was collected from ebooks from the Gutenberg Project[21]. The input ontology was developed manually using Protégé [NFM00]. This ontology is described in [SDGDG11]. This system exploits dependency parsing in the acquisition of extracting rules to be used in the ontology population process.

FRED [PDG12], LODifier [APR12] & ASKNet [HC07] These use the Discourse Representation Theory (DRT) [Kam81] to achieve the task of *Ontology Population*. They use the Boxer tool [Bos08] to get the DRT representation of the text. They are open-domain. They learn the ontology then populate it.

OwlExporter [WKR10] This system is a plug-in for the GATE system [Cun02]. It exports GATE linguistic annotations to owl format.

In the next section we propose a criteria to classify these system.

2.4.3 Classification Criteria

We propose some criteria to classify the systems presented in section 2.4.2. Some criteria is taken from [RMMGCN$^+$11].

- Elements Learnt: In the *Ontology Population*, instances of concepts and instances of relations are extracted. Some systems extract only instances of concepts like [MC06] and Ontopop [TL12] and some systems extract only instances of relations like ontoX [YM07] and [DBVSW07]. Some systems extract both like *Ruiz-Martínez et al.* [RMMGCN$^+$11].

[18] http://hermit-reasoner.com/
[19] http://nlp.stanford.edu/software/lex-parser.html
[20] http://www.lonelyplanet.com/
[21] http://www.gutenberg.us/

- **Degree of Automation**: Some systems need the intervention of the user or the domain expert like ontoX [YM07]. Some other systems are fully automated like *Ruiz-Martínez et al.* [RMMGCN+11].
- **Domain Portability**: Some systems were designed to work with a specific domain and can not be applied on other domains like *Faria and Girardi* [FG11] for the FamilyLaw domain. Some systems are open-domain like *Ruiz-Martínez et al.* [RMMGCN+11], FRED [PDG12], LODifier [APR12] and ASKNet [HC07].
- **Consistency Maintenance**: *Ontology Population* needs *reasoning* to maintain the ontology consistency during the population process. SOBA [BCRS06] and [MNS03] do reasoning but Ontopop [TL12] and *Faria and Girardi* [FG11] do not.
- **Language Dependency**: Most of the systems were built for the English language like [Emb04]. Some other systems were built for German language like [MNS03] and others for Spanish like *Ruiz-Martínez et al.* [RMMGCN+11]. The systems which use NLP can be adapted to work with different languages as NLP can be adapted for different languages.
- **The Techniques Used**:

  SOBA [BCRS06], KIM [PKO+04], *Ruiz-Martínez et al.* [RMMGCN+11], *Faria and Girardi* [FG11], Ontopop [TL12] and *Sadoun et al.* [SDGDG13] use NLP techniques.

  FRED [PDG12], LODifier [APR12] and ASKNet [HC07] use the Discourse Representation Theory [Kam81].

  [CV05] and [CDF+00] use Machine Learning.

Table 2.1 makes a summary of some systems according to the criteria proposed adding two columns about the domain type and the input ontology.

Table 2.1: Classification Ontology Population Systems from Text

System	Elements Learnt	Degree of Automation	Domain Portability	Consistency Maintenance	Language Dependency
SOBA [BCRS06]	Instances of concepts and relations	Automatic	Portable if customized	The Hermit reasoner	Language Dependent
Text-To-Onto [MS00]	Instances of concepts and relations	Semi-automatic	Domain-specific	OntoBroker reasoner [DEFS99]	German only
Faria and Girardi [FG11]	Instances of concepts and relations	Automatic	Domain-specific	No reasoning	English Only but could be adaptable
Ruiz-Martínez et al. [RMMGCN$^+$11]	Instances of concepts and relations	Automatic	Portable if customized	The HermiT reasoner	Language Independent
Ontopop [TL12]	Instances of concepts only	Semi-automatic	Portable if customized	No reasoning	Language independent
Sadoun et al. [SDGDG13]	Instances of concepts and relations	Automatic	Portable if customized	Jena reasoning	Language independent
FRED [PDG12], LODifier [APR12] and ASKNet [HC07]	Instances of concepts and relations	Automatic	Portable	No reasoning	Language independent if customized
OwlExporter [WKR10]	Instances of concepts and relations	Automatic	Portable	Any description logic reasoner	Language independent

Table 2.1: Classification Ontology Population Systems from Text...*Continuation*

System	The Techniques Used	Domain Type	The Ontology
SOBA [BCRS06]	NER, POS Tagging	The Sport Domain	The SWInto ontology [OAH+07]
Text-To-Onto [MS00]	Tokenization, Morphological Analysis, POS Tagging, NER, Chunk Parsing, Machine Learning	A specific domain	It bootstraps its own ontology
Faria and Girardi [FG11]	NER, POS Tagging and Morpho-lexical Analysis	The Family Law domain	An ad hoc manually constructed ontology
Ruiz-Martínez et al. [RMMGCN+11]	Tokenization, POS Tagging, Lemmatisation, Parsing and NER	E-Toursim Domain	travel.owl
Ontopop [TL12]	NER, LSA, Dependency Parsing	Tourism	The GETESS Ontology [SBB+99]
Sadoun et al. [SDGDG13]	Dependency Parsing	The smart space domain	[SDGDG11]
FRED [PDG12], LODifier [APR12] and ASKNet [HC07]	DRT	Open domain	They learn and populate
OwlExporter [WKR10]	GAPE annotations (POS Tagging, NER, Tokenization,...)	Open domain	Any ad hoc ontology

In the next section we present a system for extracting educational information.

3

A System for Extracting Educational Information

In this chapter, we present the system developed for extracting information from Learning Management Systems(LMS) into an ontology. The University of Évora Moodle is the LMS [WW07a] used for the tests. An OWL ontology was drawn to represent the information extracted from the LMS. After extraction, the learning management system will be ontology-based system which will give it more flexibility in making statistics about its content information which will be presented in chapter 4. It will also be possible to create a question answering system to answer natural language questions and this will be presented in chapter 5.

The architecture of the system is presented in figure 3.1.

In this system:

- SQL Queries: are used to retrieve the information from the moodle database.
- Ontology: is populated with the information retrieved from moodle. This ontology was constructed manually in order to represent the information of the moodle courses.

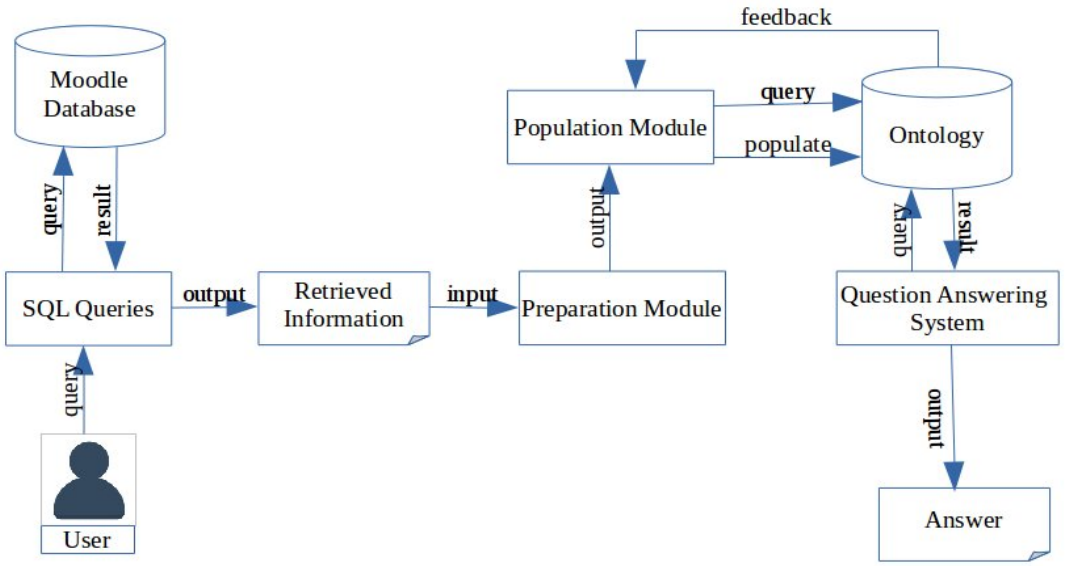

Figure 3.1: System Architecture

- Preparation Module: prepares the retrieved information to populate the ontology. the sql queries results are rewritten.
- Population Module: updates the ontology with instances of classes, object properties and data type properties.
- Question answering system: is the user interface to the ontology, it translates a user natural language query into a SPARQL query and will be presented in chapter 5. But the system enables the use of SPARQL queries directly by the users as we show in chapter 4.

Figure 3.2 shows a flowchart for extracting a specific course information. The course attributes are retrieved and its content is processed to populate the ontology with a new class instance and some data properties (id, name,..). The relations between the course and its users are retrieved (such as the users enrolled in the course and their roles, teacher, student, ..) and represented as object properties in the ontology.

In next sections we present the modules of this system.

3.1 Moodle

Moodle has a relational database with 358 tables. In this work we use the tables that contain information about: courses, users, contexts, user roles, activities, resources and user activities. The data from the University of Evora e-learning courses is used in this work. In table 3.1, some of the Moodle tables with some of their attributes are presented.

3.1.1 SQL Queries

A set of sql queries were written to retrieve the information to populate the ontology taking into account the Moodle database structure. Some of the sql queries which retrieve the information are presented. For example,

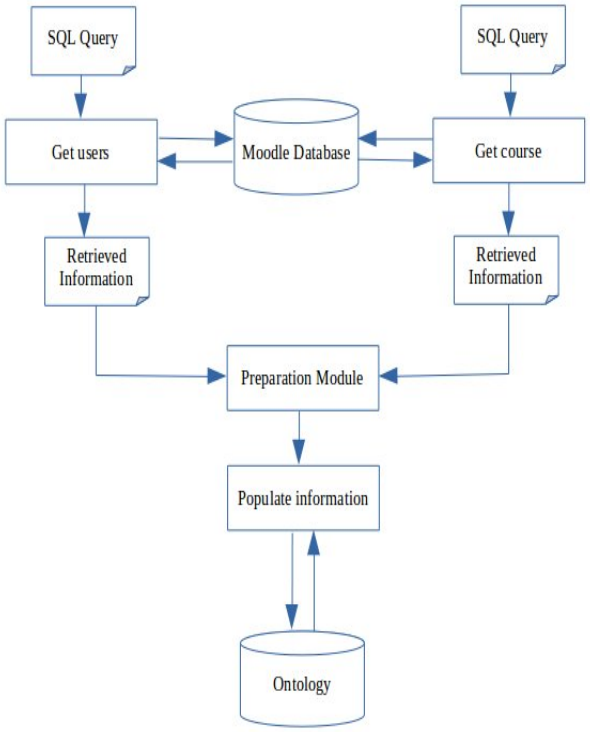

Figure 3.2: A system case to extract a specific course and its users

Table name	Columns				
mdl_course	ID	Short name	Full name	Summary	Category
mdl_user	ID	User name	First name		
mdl_role	ID	shortname	archetype		
mdl_role_assignments	ID	Role ID	Context ID	User ID	
mdl_assign	ID	Course ID			
mdl_assign_submission	ID	Assignment ID	User ID	Status	
mdl_quiz	ID	Course ID	Name		
mdl_files	ID	Source	File name	Content hash	
mdl_page	ID	Course	Name	Content	
mdl_grade_items	ID	Course ID	Item Instance	Item module	
mdl_logstore_standard_log	ID	Course ID	User ID	Activity type	
mdl_context	ID	Context level	Instance ID		
mdl_enrol	ID	Course ID			
mdl_user_enrolments	ID	Enrol ID	User ID		

Table 3.1: Some of the tables used with some of their columns

```
select c.shortname course, u.username username
  from mdl_course c, mdl_enrol e, mdl_user_enrolments ue, mdl_user u
  where
       c.id = 6 and
       e.courseid = c.id and
       e.id = ue.enrolid and
       ue.userid = u.id
```

Figure 3.3: A sql query to get all the students enrolled in a specific course

Course	Username
course1	user1
course1	user2
...	...

Table 3.2: sql query (figure 3.3) result

a sql query to get all the students which are enrolled in a specific course (course id is 6) is presented in figure3.3. This query retrieves information from four different tables related to each other by a primary key-foreign key relations. Figure 3.4 presents the relations between these four tables. For example, the table mdl_course is related to the table mdl_enrol by one-to-many relation. This query result is presented in table 3.2.

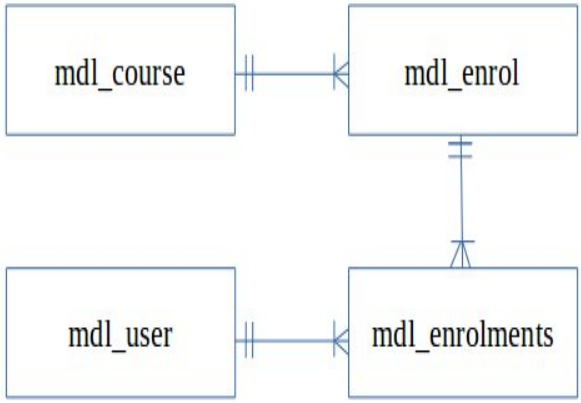

Figure 3.4: A relation diagram between the four tables

The queries can get more nested if we are looking for more specific information. The query in figure 3.5 is used to get the actions done by users according to their different roles (student, editing-teacher, non-editing teacher). This query uses four tables, one of them is the mdl_logstore table. This table records all the actions done by all the users like viewing or updating. The result of this query is presented in the table 3.3.

In figures 3.6, 3.7, 3.8 we present more sql queries:

- 3.6 to retrieve the users who took quizzes in a specific course (with id=6).
- 3.7 to retrieve all the users who passed a specific quiz in a specific course (course id=6, quiz id=19).
- 3.8 to retrieve all the files for a specific course (course id=5).

```
select r.shortname as role, l.action, count( l.userid ) as usersNumber
  from mdl_logstore_standard_log as l
  join mdl_context as context on
       context.instanceid = l.courseid
       and context.contextlevel = 50
  join mdl_role_assignments as ra on
       l.userid = ra.userid
       and ra.contextid = context.id
  join mdl_role as r on
       ra.roleid = r.id
  where ra.roleid in ( 3,4,5 )
  group by roleid, l.action, r.shortname
```

Figure 3.5: A sql query to get the actions of the users with different roles

Role	Action	UsersNumber
student	updated	num1
student	created	num2
editingteacher	viewed	num3
...	...	...

Table 3.3: sql query (figure 3.5) result

```
select u.username
  from
       mdl_quiz q, mdl_quiz_attempts qa, mdl_user u, mdl_course c
  where
       q.id = qa.quiz and qa.userid = u.id and qa.state = 'finished'
       and q.course = c.id and c.id = 6
```

Figure 3.6: A sql query to get all the users which took quizzes in a specific course

```
select u.username
  from
       mdl_quiz q, mdl_quiz_attempts qa, mdl_user u, mdl_course c
  where
       q.id = qa.quiz and qa.userid = u.id and qa.state = 'finished'
       and q.course = c.id and c.id = 6 and q.id = 19 and
       qa.sumgrades >=
         (select gi.gradepass
            from
                 mdl_course c, mdl_quiz q, mdl_grade_items gi
            where
                 c.id = q.course and c.id = gi.courseid
                 and q.id = gi.iteminstance
                 and c.id = 6 and q.id = 19 and
                 gi.itemmodule = 'quiz'
         )
```

Figure 3.7: A sql query to get all the users who passed a specific quiz in a specific course

```
select
    mdl_files.source,
    mdl_files.filename,
    mdl_context.path,
    mdl_resource.course,
    mdl_resource.name,
    mdl_course.fullname,
    mdl_course.shortname
  from
    mdl_files
    inner join mdl_context on mdl_files.contextid = mdl_context.id
    inner join mdl_resource on mdl_context.instanceid = mdl_resource.id
    inner join mdl_course on mdl_resource.course = mdl_course.id
  where (mdl_course.id = 5)
```

Figure 3.8: A sql query to get all the files of a specific course

Class	Object Property	Domain	Range
Course	hasActivity	Course	CourseActivity
	hasUser	Course	User
User	isStudentOf	User	Course
	isTeacherOf	User	Course
	doActivity	User	CourseActivity
UserActivity	activityByUser	UserActivity	User
	activityInCourse	UserActivity	Course

Table 3.4: Ontology classes with their object properties

3.2 Ontology

In this work, the ontology was built manually and inspired from the structure of the moodle database. It was built using protégé [NFM00]. Moodle database tables give rise to classes or object properties. The attributes of the tables give rise to data type properties or object properties.

Table 3.4 shows some of the ontology classes and their object properties. “Course”, “User” and “UserActivity” are classes. The object properties are relations between the classes such that each object property has a domain class and a range class. For example, the “hasActivity” object property has the domain “Course” and the range “CourseActivity”. That means each course (each instance of the class “Course”) in the ontology can be related to one or more activities which are instances from the class “CourseActivity”.

In table 3.5 we present some data type properties. Data type properties are relations between classes and values. For example, the data type property “CourseID” is a property of the class “Course” (has this class as a domain) which means that each instance of the class “Course” can have a “CourseID” property. This property has the range “int” which means that its value should be an integer number.

Figure 3.9 presents a part of the ontology structure using the OntoGraf utility of protégé. It contains some of the classes of the ontology and the arrows between them refer to the object properties between these classes. The arrow starts from the domain class and goes to the range class.

Some examples of the ontology classes are:

- Each course in the moodle has activities and resources. Activities in moodle are things like quizzes, as-

Class	Data Property	Domain	Range
Course	Course ID	Course	xsd:int
	Full name	Course	xsd:string
User	User ID	User	xsd:int
	First name	User	xsd:string
	Last name	User	xsd:string
UserActivity	Activity type	UserActivity	xsd:string
	Week number	UserActivity	xsd:int

Table 3.5: Ontology classes with their data properties

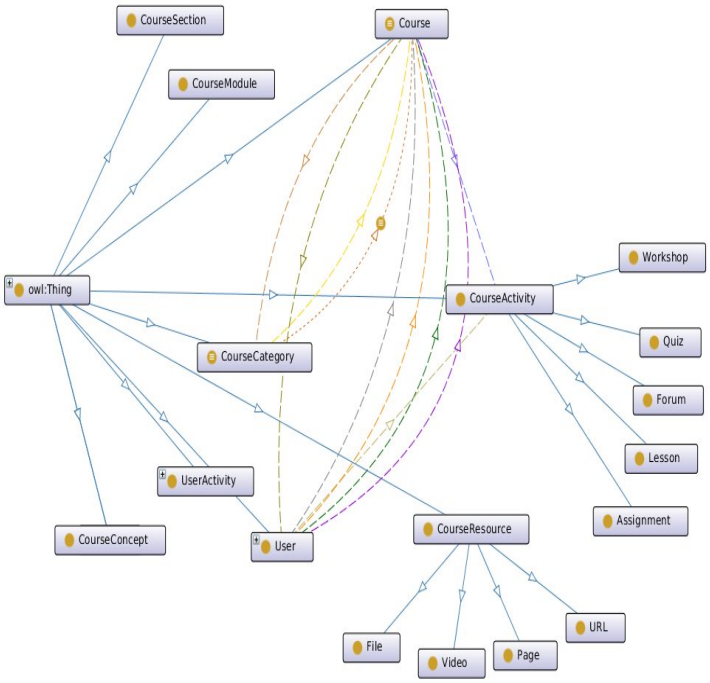

Figure 3.9: The Ontology View, OntoGraf

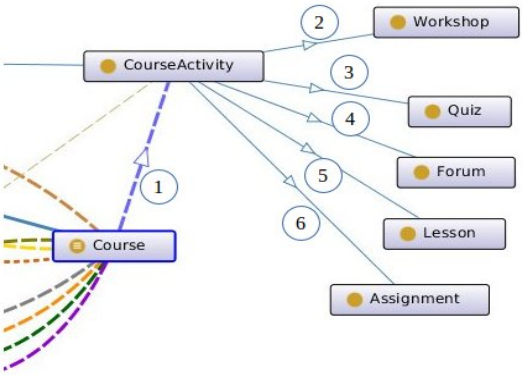

Figure 3.10: The Course and CourseActivity relation

signments, forums, etc; resources are things like URLs, files, etc. The ontology will have classes and subclasses to represent the information retrieved from moodle about course, activities and resources and relationships between them. The ontology has the class “CourseActivity” which has subclasses to represent the different types of the course activities. The class “Course” is related to the class “CourseActivity” by an object property “hasActivity”. Figure 3.10 presents how the two classes are related. The arrow with the number 1 represents the “hasActivity” object property. The direction of the arrow refers to the domain of this relation which is the “Course” class and the range of it which is the “CourseActivity” class. The arrows with the numbers 2, 3, 4, 5, 6 represent the “hasSubclass” or “isA” object property. This refers that the classes “Workshop”, “Quiz”, “Forum”, “Lesson”, “Assignment” are subclasses of the class “CourseActivity”. Having this object property in the ontology with the these subclasses make it possible to know the activities of a specific course with their types.

- The “User” class represents the user and the “UserActivity” class represents the activities of the user in the course like viewing or deleting some thing in this course. They are related by object properties like “activityByUser” which represents the user who does the activity. The object property “activityInCourse” represents in which course the activity took place. The object property “hasUser” refers that a specific user is enrolled in a specific course. 3.11 shows how these classes are related. The arrow with the number 1 represents the “activityByUser” object property. The arrow with the number 2 represents the “activityInCourse” object property. The arrow with the number 3 represents the “hasUser” object property. Having these three object properties in the ontology make it possible to know which user does a specific activity in a specific course.

3.3 Preparation Module

This module processes the sql results to create triples to be added to the ontology. For example, to populate a course into the ontology, this module prepare triples with the course id, its short name and its duration. These triples, (Course, id, short name), (Course, id, course duration) are the input of the population module where they will be used to populate the ontology.

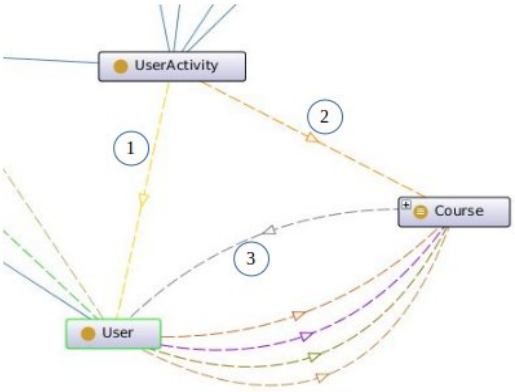

Figure 3.11: The Course, User and UserActivity relations

Number of classes	Number of instances of classes
25	26234
Number of object properties	Number of data type properties
15	37

Table 3.6: Ontology statistics after population

3.4 Population Module

This module will populate the ontology with the triples produced by the preparation module. For example, figure 3.12 shows the flowchart diagram of adding two triples, first one is a course instance with its id and short name and the second one is a user activity which took place in this course with its id and type. The course should be added as an instance of the class “Course” in the ontology and the user activity as an instance of the class “UserActivity”. Then the “activityInCourse” object property between the two individuals should be added to say that this activity took place in this course. It is always checked for the individual existence in the ontology before adding it. In jena API, when adding an individual which has the same name of an already existed individual, the latter will smash the former. This causes some loss of information as some individuals may have the same name but they may be distinct objects.

Another example is presented in figure 3.13. The ontology is populated with new instances of the classes “Course” and “User” and the relations (object properties) between these instances. Users can have different roles in the course like student, teacher, editing teacher, etc. If the user is a student in the course then the “isStudentOf” object property is added between the two instances. If the user is a teacher in the course then the “isTeacherOf” object property is added between the two instances.

Table 3.6 shows some statistics about the ontology after population:

The ontology will be prepared to be queried using sparql to answer questions about its content. This will be presented in the next chapters.

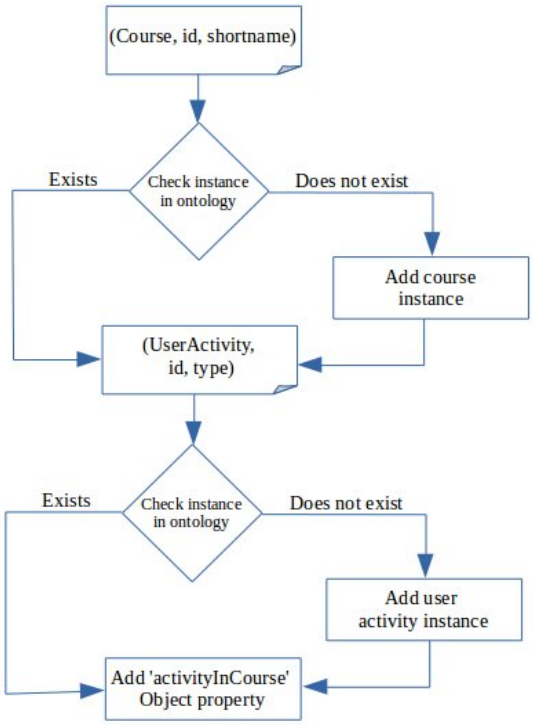

Figure 3.12: Adding two instances and an object property to the ontology

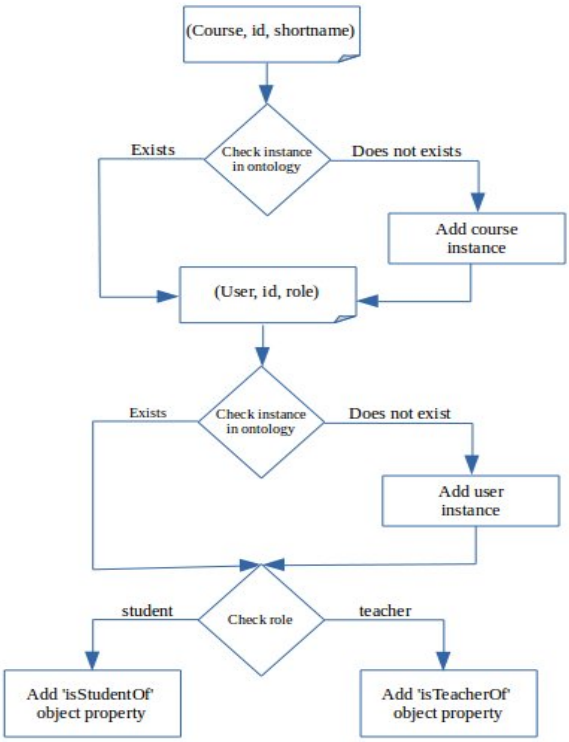

Figure 3.13: Adding users and their roles in a specific course

4

Learning Analysis

In this chapter we present some examples of how the populated ontology can be exploited to get information and statistics about its content. The ontology is queried using SPARQL query language. The Java JENA API is used to run SPARQL queries.

The SPARQL result is then processed to visualize the answer as a chart. The JFreeChart[1] Java chart library is used to achieve that. The architecture of this system is presented in Figure 4.1.

In this system:

- SPARQL Query: SPARQL query is the query that the user writes.

  An example of a SPARQL query to get all the instances of the Course class from the ontology is presented in figure 4.2. This query has rdf prefix which is a standard name space and ns prefix which is the chosen name space of the ontology. ?x is a variable of the type Course.

- Query the ontology: Java Jena API is used to query the ontology using SPARQL.

  Jena code presented in figure 4.3. queryString is the SPARQL query. ontModel is the ontology model.

[1] http://www.jfree.org/jfreechart/

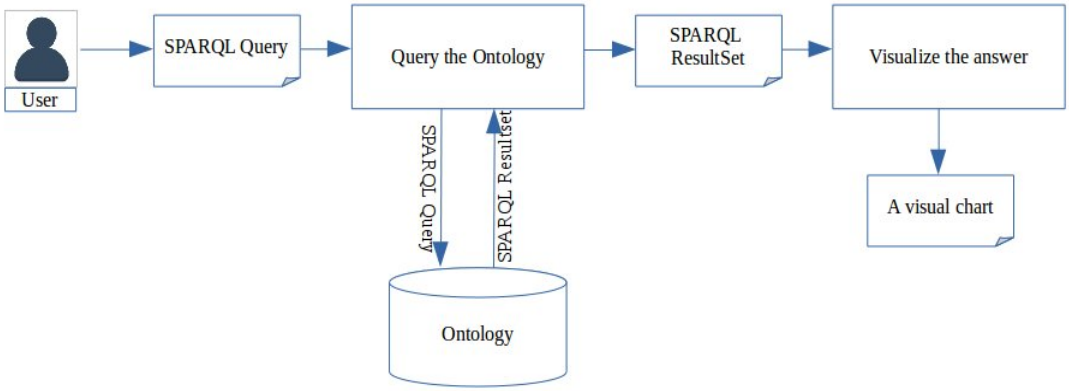

Figure 4.1: The Learning Analysis System Architecture

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns: <http://www.semanticweb.org/courses-ontology-57#>
 select ?x
  where {
         ?x rdf:type ns:Course.
        }
```

Figure 4.2: A SPARQL query example

```
Query query = QueryFactory.create(queryString);
QueryExecution qe = QueryExecutionFactory.create(query, ontModel);
com.hp.hpl.jena.query.ResultSet results =  qe.execSelect();
```

Figure 4.3: Jena code to query the ontology using SPARQL

```
{
  "head": {
    "vars": [ "x" ]
  } ,
  "results": {
    "bindings": [
      {
      "x": { "type": "uri" , "value": "http://www.semanticweb.org/courses-ontology-57#COURSE1" }
      } ,
      {
      "x": { "type": "uri" , "value": "http://www.semanticweb.org/courses-ontology-57#COURSE2" }
      } ,
      {
      "x": { "type": "uri" , "value": "http://www.semanticweb.org/courses-ontology-57#COURSE3" }
      } ,
      {
      "x": { "type": "uri" , "value": "http://www.semanticweb.org/courses-ontology-57#COURSE4" }
      } ,...
```


Figure 4.4: The result of SPARQL query in json format

- SPARQL result set: it is obtained and stored in a json file. For example, the result of the query (see figure 4.2) is presented in figure 4.4.
- Visualize the answer: it will process the json file and present a graph as a result. Java JFreeChart chart library is used to achieve that. This will be presented in next sections.
- A visual chart: the visual result. Examples are given in the next sections.

Three use cases of this system are demonstrated in the next sections.

4.1 Use Case 1

Use case 1 is an example of how to retrieve statistics about all the courses in the ontology. These statistics are:

- The number of activities in the course
- The number of resources in the course
- The number of the reading activities of the students in the course
- The number of the reading activities of the teachers in the course
- The number of the submission activities of the students in the course
- The number of the submission activities of the teachers in the course
- The number of students registered in siiue in the course
- The number of students of the course
- The number of the teachers of the course

- The number of students who passed the course

The SPARQL query in figure 4.5 is used to retrieve this information.

This query is composed by a first part where the rdfs prefix is standard prefix and in the where clause we have:

- (1), (9), (11), (13) and (15) define variables of the classes Course, User and Grade.
- (10), (12), (14), (16) and (17) define object property constraints.
- (2), (3), (4), (5), (6), (7), (8) and (18) define data type property constraints.
- The keyword “optional” merges all the subqueries when the variable course is equal for all.

The json result (only one entry) of this SPARQL query is presented in figure 4.6.

In tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, SPARQL query result is presented.

Course	1	2	3	4	5	6	7	8	9	10
#Activities	14	2	9	8	13	10	2	19	15	3
#Resources	13	31	57	38	67	8	14	103	42	24
#Read of students	1160	6216	828	1296	5642	1406	799	12415	5131	788
#Read of teachers	578	3538	336	420	1518	633	233	3645	1687	340
#Submissions of students	112	358	67	236	1004	204	61	4166	472	21
#Submissions of teachers	207	836	398	245	618	136	127	2259	453	164
#Students in SIIUE	4	11	4	33	36	3	34	36	15	4
#Students of course	3	24	3	12	26	3	9	46	9	4
#Teachers of course	2	5	1	1	3	1	1	3	1	1
#Students passed the course	3	20	2	0	13	3	0	0	6	3

Table 4.1: Statistics of the courses

Course	11	12	13	14	15	16	17	18	19	20
#Activities	19	16	19	3	2	10	18	11	20	10
#Resources	33	54	54	24	5	56	27	84	47	16
#Read of students	3693	3270	1830	455	254	4666	1870	2280	9350	4155
#Read of teachers	1760	913	358	167	251	1210	738	1209	3121	1140
#Submissions of students	512	442	179	10	37	214	175	214	921	410
#Submissions of teachers	1118	448	342	45	65	452	485	310	2146	462
#Students in SIIUE	30	30	34	5	6	12	3	2	31	15
#Students of course	17	18	8	5	5	11	3	2	22	15
#Teachers of course	1	2	1	1	1	1	1	1	1	1
#Students passed the course	3	0	0	3	5	11	3	2	9	7

Table 4.2: Statistics of the courses

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns: <http://www.semanticweb.org/courses-ontology-57#>
 select *
  where {
         {
           select *
            where{
                  ?course rdf:type ns:Course.                        (1)
                  ?course ns:courseActivityNum ?courseActivityNum.   (2)
                  ?course ns:courseResourcesNum ?courseResourcesNum. (3)
                  ?course ns:studentReadNum ?studentReadNum.         (4)
                  ?course ns:teacherReadNum ?teacherReadNum.         (5)
                  ?course ns:studentSubmissionsNum ?studentSubNum.   (6)
                  ?course ns:teacherSubmissionsNum ?teacherSubNum.   (7)
                  ?course ns:studentsSIIUENum ?studentsSIIUENum.     (8)
                 }
        }
           optional
        {
           select ?course (count(?student) as ?studentsNum)
            where{
                  ?student rdf:type ns:User.        (9)
                  ?student ns:isStudentOf ?course.  (10)
                 } group by ?course
        }
           optional
        {
           select ?course (count(?teacher) as ?teachersNum)
            where{
                  ?teacher rdf:type ns:User.         (11)
                  ?teacher ns:isTeacherOf ?course.   (12)
                 } group by ?course
        }
           optional
        {
           select ?course (count(?student) as ?passedStudents)
            where{
                  ?student rdf:type ns:User.          (13)
                  ?student ns:isStudentOf ?course.    (14)
                  ?grade rdf:type ?Grade.             (15)
                  ?grade ns:gradeInCourse ?course.    (16)
                  ?grade ns:gradeObtainedBy ?student. (17)
                  ?grade ns:gradePassed 'S'           (18)
                 } group by ?course

        }}
```

Figure 4.5: Statistics of all the courses

```
{
  "head": {
    "vars": [ "course" , "courseActivityNum" , "courseResourcesNum" ,
              "studentReadNum" , "teacherReadNum" , "studentSubmissionsNum" ,
              "teacherSubmissionsNum" , "studentsSIIUENum" , "studentsNum" ,
              "teachersNum" , "passedStudents" ]
  } ,
  "results": {
    "bindings": [
      {
     "course": { "type": "uri" , "value": "http://www.semanticweb.org/courses-ontology-57#COURSE" } ,
       "courseActivityNum": { "type": "literal" , "value": "14" } ,
       "courseResourcesNum": { "type": "literal" , "value": "13" } ,
       "studentReadNum": { "type": "literal" , "value": "1160" } ,
       "teacherReadNum": { "type": "literal" , "value": "578" } ,
       "studentSubmissionsNum": { "type": "literal" , "value": "112" } ,
       "teacherSubmissionsNum": { "type": "literal" , "value": "207" } ,
       "studentsSIIUENum": { "type": "literal" , "value": "4" } ,
       "studentsNum": { "datatype": "http://www.w3.org/2001/XMLSchema#integer" ,
                        "type": "typed-literal" , "value": "3" } ,
       "teachersNum": { "datatype": "http://www.w3.org/2001/XMLSchema#integer" ,
                        "type": "typed-literal" , "value": "2" } ,
       "passedStudents": { "datatype": "http://www.w3.org/2001/XMLSchema#integer" ,
                           "type": "typed-literal" , "value": "3" }
     } , ...
```


Figure 4.6: The result of SPARQL query

Course	21	22	23	24	25	26	27	28	29	30
#Activities	15	39	12	10	11	14	10	12	1	12
#Resources	81	107	1	48	16	14	60	50	4	17
#Read of students	5086	6409	1715	902	2466	429	5625	4231	5281	2482
#Read of teachers	1262	4105	265	1273	502	457	1623	1221	2674	450
#Submissions of students	1345	1760	142	46	193	7	2298	322	740	211
#Submissions of teachers	1090	1264	106	893	261	115	753	471	1029	242
#Students in SIIUE	28	12	30	6	30	5	6	29	26	15
#Students of course	12	14	13	4	9	4	7	18	19	11
#Teachers of course	1	1	1	1	1	1	1	1	1	1
#Students passed the course	0	0	0	3	12	4	3	0	3	8

Table 4.3: Statistics of the courses

Course	31	32	33	34	35	36	37	38	39	40
#Activities	16	11	3	25	1	14	19	19	13	7
#Resources	27	27	22	31	10	22	103	52	48	56
#Read of students	1518	2279	1374	1910	736	1896	3540	4216	9317	2029
#Read of teachers	1176	417	865	514	360	443	1002	1243	1012	548
#Submissions of students	192	205	106	177	72	140	1201	410	1279	197
#Submissions of teachers	681	460	289	246	72	191	644	703	518	507
#Students in SIIUE	4	33	3	34	5	3	36	29	30	34
#Students of course	4	23	3	6	4	3	17	22	23	11
#Teachers of course	1	1	1	1	1	1	5	1	1	1
#Students passed the course	3	5	3	0	3	3	0	8	10	0

Table 4.4: Statistics of the courses

Course	41	42	43	44	45	46	47	48	49	50
#Activities	22	9	3	15	2	23	8	7	12	9
#Resources	64	1	23	4	7	37	29	19	2	19
#Read of students	1917	2844	1816	996	477	1581	1055	3353	2027	1378
#Read of teachers	672	831	471	604	252	333	629	1089	257	588
#Submissions of students	269	352	128	83	15	137	111	500	116	99
#Submissions of teachers	386	287	202	158	61	209	121	779	159	343
#Students in SIIUE	33	16	15	4	4	34	3	30	30	2
#Students of course	9	9	9	5	4	8	3	20	24	2
#Teachers of course	1	1	1	1	2	1	1	2	1	1
#Students passed the course	0	6	7	3	4	0	3	0	3	2

Table 4.5: Statistics of the courses

Course	51	52	53	54	55	56	57	58	59	60
#Activities	24	9	15	19	13	21	13	13	1	11
#Resources	80	18	67	35	69	23	29	78	15	53
#Read of students	5726	1409	1222	5453	1823	5650	3869	3113	324	2891
#Read of teachers	1631	488	452	2899	488	2730	763	1554	146	1466
#Submissions of students	335	173	52	393	267	809	445	262	14	274
#Submissions of teachers	795	134	311	1013	422	1603	259	1227	57	578
#Students in SIIUE	15	3	4	15	34	31	15	28	5	4
#Students of course	14	3	4	13	5	10	9	11	5	4
#Teachers of course	1	1	1	1	1	1	1	1	4	1
#Students passed the course	8	3	2	6	3	0	6	2	0	4

Table 4.6: Statistics of the courses

In figures 4.7, 4.8, 4.9, 4.10, 4.11, 4.12 we present the table as a chart. These figures are generated using JFreeChart library after processing the json file result. Two scales are created due to the differences in the numbers values.

In the graph legend, the labels marked with an asterisk belong to the second scale. The horizontal axis represents the courses.

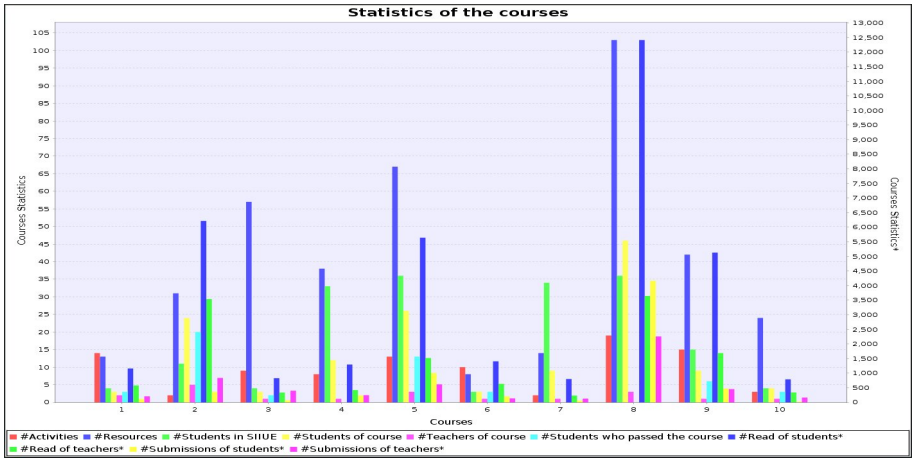

Figure 4.7: Statistics of the courses

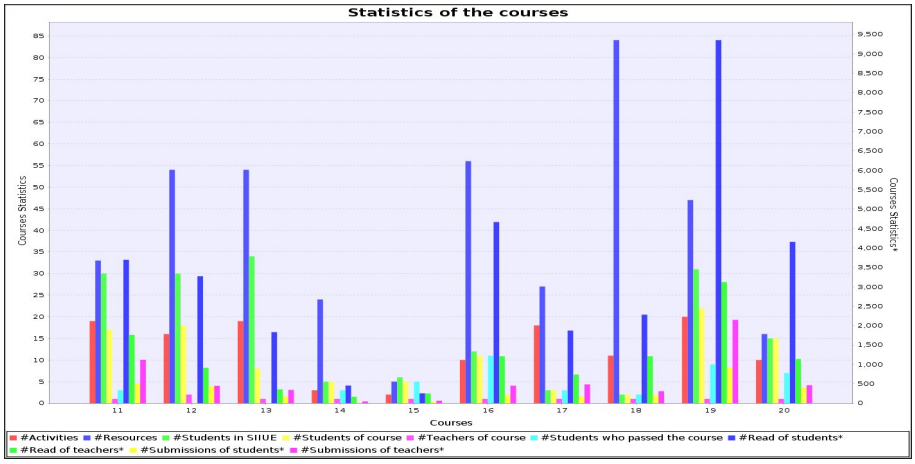

Figure 4.8: Statistics of the courses

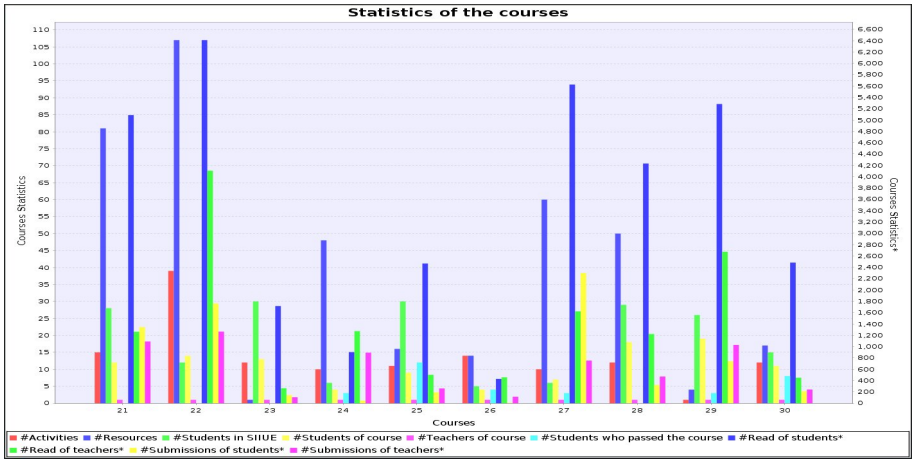

Figure 4.9: Statistics of the courses

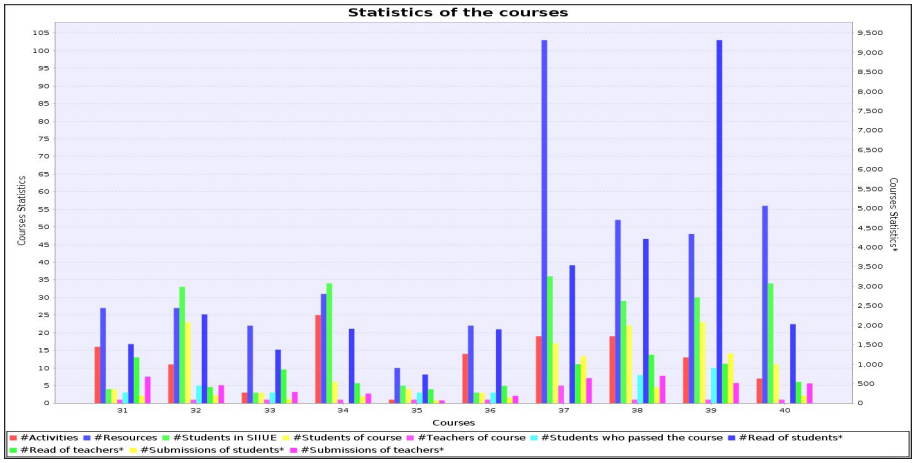

Figure 4.10: Statistics of the courses

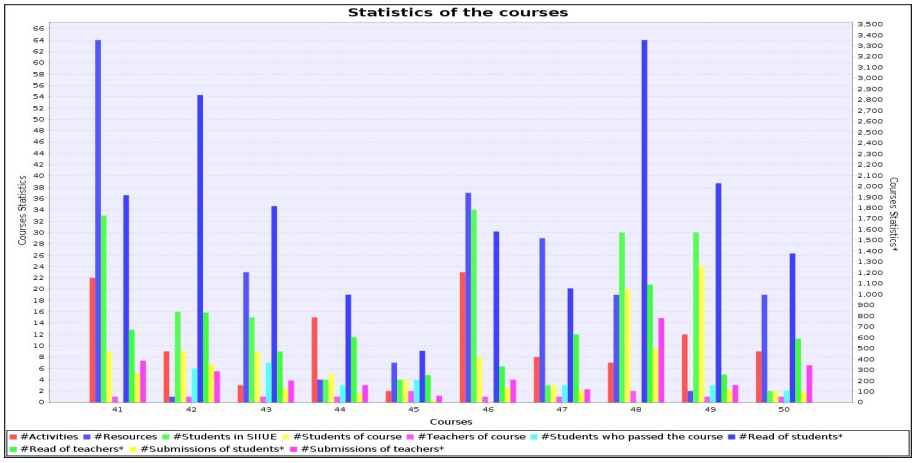

Figure 4.11: Statistics of the courses

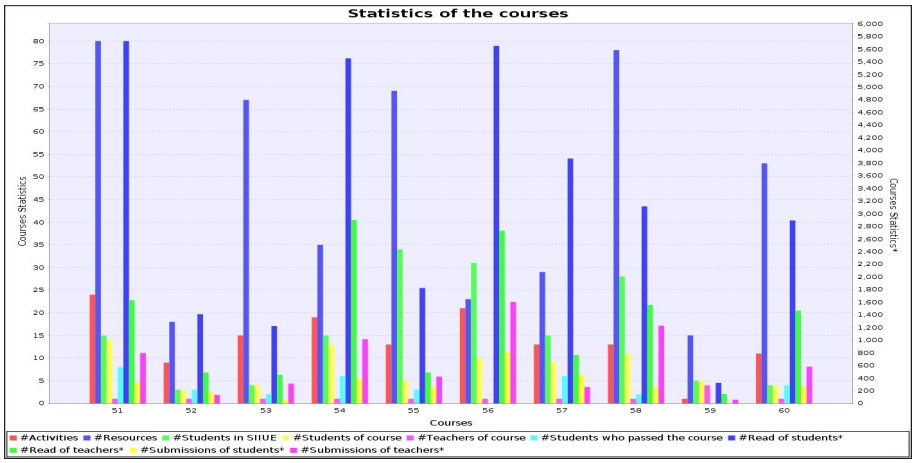

Figure 4.12: Statistics of the courses

These figures provide a general view about all the courses and an easy way to look at their characteristics (number of activities, resources or students...) and understand the differences between them to make some conclu-

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns: <http://www.semanticweb.org/courses-ontology-57#>
 select ?user ?week ?grade ?gValue ?gPassed (sum(?num) as ?numActWeek)
  where {
         ?user rdf:type ns:User.                        (1)
         ?course rdf:type ns:Course.                    (2)
         ?userActivity rdf:type ns:UserActivity.        (3)
         ?userActivity ns:activityInCourse ?course.     (4)
         ?user ns:isStudentOf ?course.                  (5)
         ?userActivity ns:activityByUser ?user.         (6)
         ?userActivity ns:week ?week.                   (7)
         ?userActivity ns:num ?num.                     (8)
         ?course ns:c_id '1545'.                        (9)
         ?grade rdf:type ns:Grade.                      (10)
         ?grade ns:gradeObtainedBy ?user.               (11)
         ?grade ns:gradeInCourse ?course.               (12)
         ?grade ns:gradeValue ?gValue.                  (13)
         ?grade ns:gradePassed ?gPassed.                (14)
        } group by ?user ?week ?grade ?gValue ?gPassed
          order by asc(UCASE(str(?user)))
```

Figure 4.13: The activities by week of students in course 'x'

sions. For example, the courses 6 and 7 have a similar number of activities and resources (18 and 16) respectively and they have (3 and 9) students respectively. The number of reads of students in course 7 (which has the bigger number of students) is less than the number of reads of course 6 (which has less students). We can conclude that the students in the course 6 were more active than in course 7 and we see than no one passed in the course 7 but all the students in course 6 passed the course.

4.2 Use Case 2

Some questions which are not possible to answer by the system which is based on a relational database, can be possible to answer using the ontology-based system.

In use case 2 we present an example for finding all the activities of the student by week and their grades in a specific course.

For a course "x" with the id=1545, by the ontology-based system, this question can be answered using the SPARQL query presented in figure 4.13. To explain this query:

- (1), (2), (3) and(10) define variables of the classes User, Course, UserActivity and Grade respectively.
- (4), (5), (6), (11), (12) define object property constraints.
- (7), (8), (9), (13), (14) define data type property constraints.

Table 4.7 shows the result of SPARQL query in figure 4.13.

	Student	1	2	3	4	5	6	7	8	9
Number of students activities by week	Week1	12	14	10	0	6	2	3	0	0
	Week2	2	14	26	0	10	39	4	0	1
	Week3	4	0	10	0	17	7	1	0	0
	Week4	3	3	21	0	6	0	9	0	2
	Week5	10	0	24	0	2	2	2	0	0
	Week6	6	7	29	0	0	2	0	0	0
	Week7	45	0	52	0	0	2	14	4	0
	Week8	32	1	20	1	31	29	0	3	0
	Week9	4	0	11	0	3	5	82	0	0
	Week10	8	8	18	0	7	22	11	0	0
	Week11	6	24	26	0	0	0	5	0	0
	Week12	39	0	7	0	1	4	1	0	0
	Week13	18	0	8	0	1	23	37	0	2
	Week14	8	36	21	0	31	61	3	0	0
	Week15	25	3	5	0	2	0	3	0	0
	Week16	8	0	13	0	1	3	24	0	0
	Week17	8	29	8	0	2	7	4	1	0
	Week18	6	2	16	0	1	2	40	0	0
	Week19	4	1	25	0	0	16	4	1	0
	Week20	7	10	8	0	1	6	10	0	0
	Week21	5	1	13	0	1	3	28	1	1
	Week22	7	22	18	0	0	4	14	0	0
	Week23	5	0	11	0	5	1	8	1	0
	Week24	5	4	12	0	9	0	11	2	0
	Week25	8	0	17	0	0	7	16	0	0
	Week26	27	0	42	0	1	11	10	3	0
	Week27	5	12	33	0	19	9	0	0	0
	Week28	8	0	16	0	16	18	10	0	0
	Week29	4	9	10	0	9	8	2	0	0
	Week30	12	0	6	0	3	2	5	3	0
	Grade	16	16	17	0	15	14	14	0	14

Table 4.7: SPARQL result for course x

Figure 4.14 visualizes the result of this query. It has two scales: first scale is for the number of activities and second one is for the grade. The horizontal axis represents the students of the course. It is observed from this graph that the more the activities the better the grade. It is also observed that students tend to have more activities in the first weeks than the last weeks.

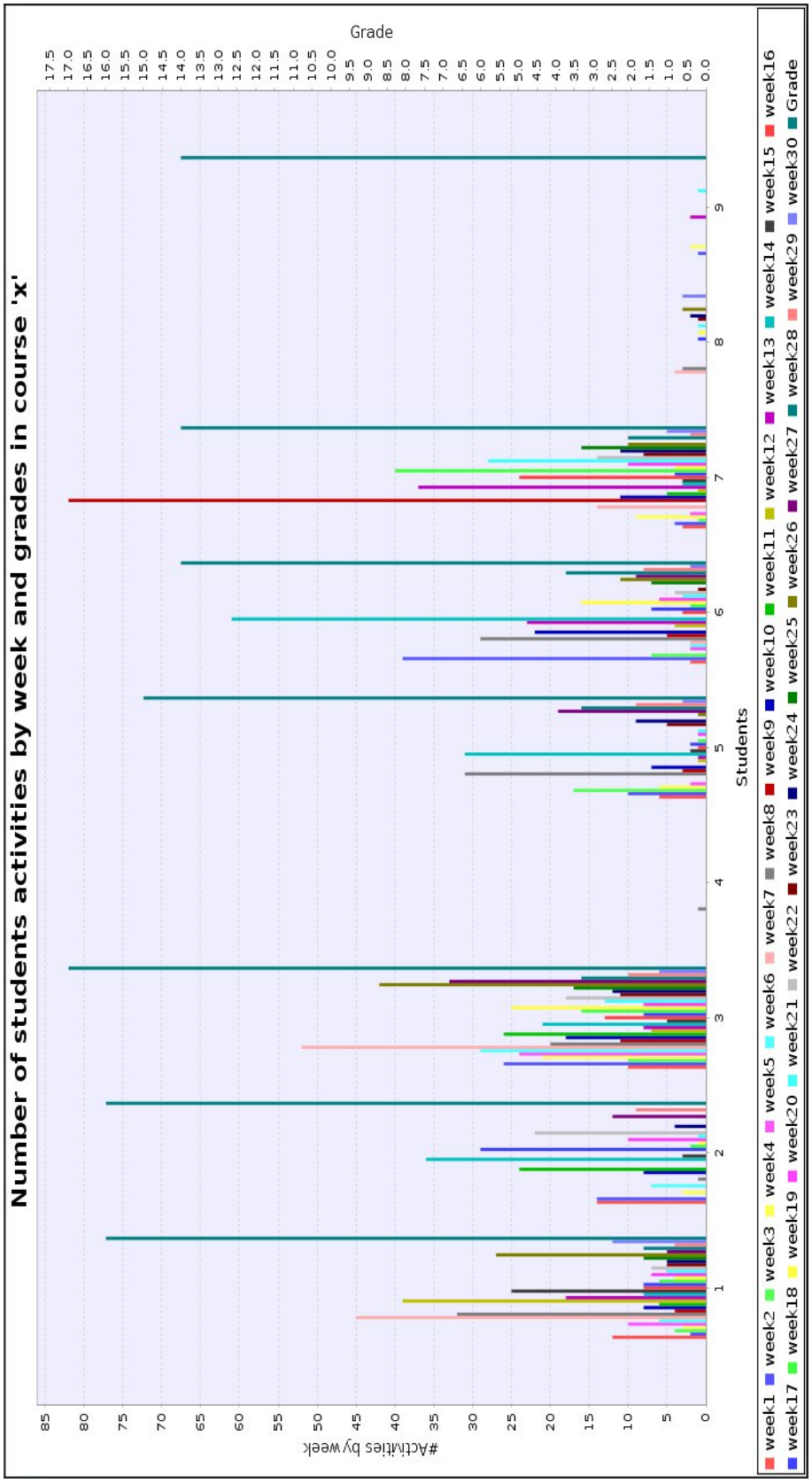

Figure 4.14: The activities by week of the students in course 'x'

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns: <http://www.semanticweb.org/courses-ontology-57#>
 select ?user ?gValue ?gPassed (sum(?num) as ?numAct)
  where {
         ?user rdf:type ns:User.                        (1)
         ?course rdf:type ns:Course.                    (2)
         ?course ns:c_id '1545'.                        (3)
         ?userActivity rdf:type ns:UserActivity.        (4)
         ?grade rdf:type ns:Grade.                      (5)
         ?userActivity ns:activityInCourse ?course.     (6)
         ?user ns:isStudentOf ?course.                  (7)
         ?userActivity ns:activityByUser ?user.         (8)
         ?userActivity ns:num ?num.                     (9)
         ?grade ns:gradeObtainedBy ?user.               (10)
         ?grade ns:gradeInCourse ?course.               (11)
         ?grade ns:gradeValue ?gValue.                  (12)
         ?grade ns:gradePassed ?gPassed.                (13)
        } group by ?user ?gValue ?gPassed
          order by asc(UCASE(str(?user)))
```

Figure 4.15: Correlation(activities-grades) in course 'x'

Student	1	2	3	4	5	6	7	8	9
#Activities	341	200	536	1	185	295	361	19	6
Grade	16	16	17	0	15	14	14	0	14
Correlation	0.69								

Table 4.8: SPARQL result for course 'x'

4.3 Use Case 3

Consider the example:

- I want to know the correlation between the number of activities each user does in a specific course and his grade in this course

For a course "x" with the id=1545, this question can be answered using the SPARQL query presented in figure 4.15. To explain this query:

- (1), (2), (4) and (5) define variables of the classes User, Course, UserActivity and Grade respectively.
- (6), (7), (8), (10) and (11) define object property constraints.
- (3), (9), (12) and (13) define data type property constraints.

Table 4.8 shows the result of SPARQL query of figure 4.15

Figure 4.16 shows the information of the table 4.8 The first scale in the chart is for the number of activities and the second scale is for the grade. The horizontal axis represents the students of the course. The positive value of the correlation indicates that the two variables are positively related. Which means, when more activities done

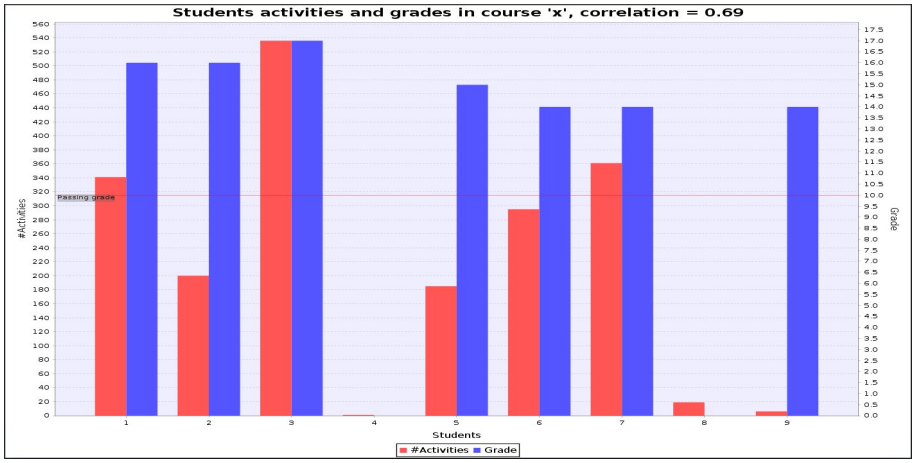

Figure 4.16: The correlation (activities, grades) of students in course 'x'

by a student then his grade is better.

Consider another course "y" with the id = 919.
Table 4.9 shows the result of SPARQL query of figure 4.15 after changing the line (3) to:

```
?course ns:c_id '919'.
```

Student	1	2	3	4	5	6	7	8
#Activities	69	726	2190	243	511	781	1458	203
Grade	0	13	12	10	10	0	12	0
Student	9	10	11	12	13	14	15	16
#Activities	95	1437	331	1183	352	137	6	51
Grade	0	14	11	18	11	0	0	0
Correlation	0.66							

Table 4.9: SPARQL result for course 'y'

Figure 4.17 shows the information of the table 4.9.

In this course also, the positive correlation says that students with more activities have better grades.

In the next chapter we explain how to convert natural language questions to SPARQL using Natural Language Processing tools and we propose an algorithm for the conversion process.

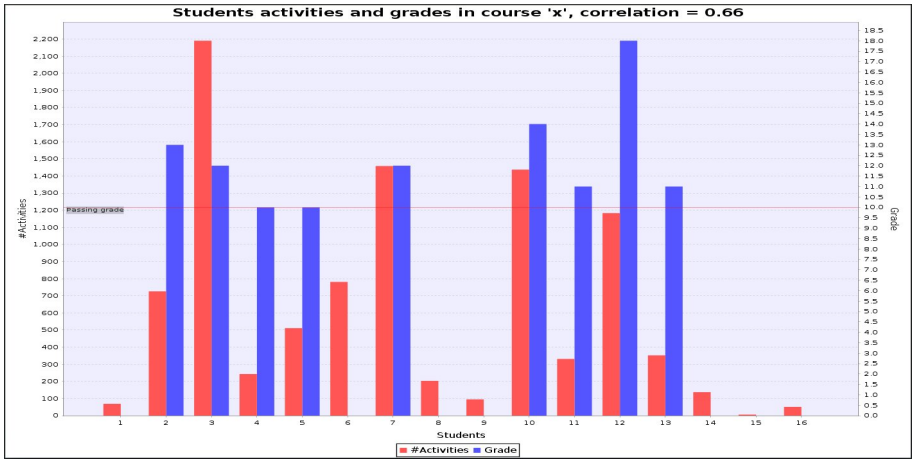

Figure 4.17: The correlation (activities, grades) of students in course 'y'

5

Question Answering System

This system was created to convert natural language questions to SPARQL queries. The structure of this system is presented in figure 5.1.

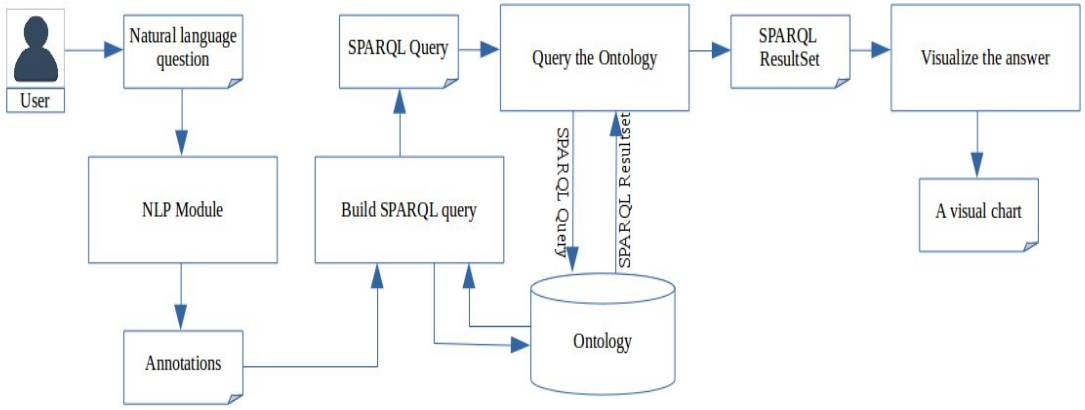

Figure 5.1: Question Answering System Architecture

The “NLP Module” processes the natural language question and produces annotations as a result. The “Build SPARQL query” module takes the annotations generated by the previous module and generates the SPARQL query. These two modules are explained in the next sections in this chapter.

5.1 NLP Module

Natural language questions are processed using the natural language tools in GATE [Cun02]. GATE is a natural language processing framework that includes tools to produce annotations for the sentence tokens. Stanford POS tagger and stanford dependency parser are included in this framework as tools. The annotations are passed into a JAPE transducer to be stored in a file. The algorithm which generates SPARQL uses these annotations.
Figure 5.2 shows the processing pipeline, each process is a tool included in the GATE framework:

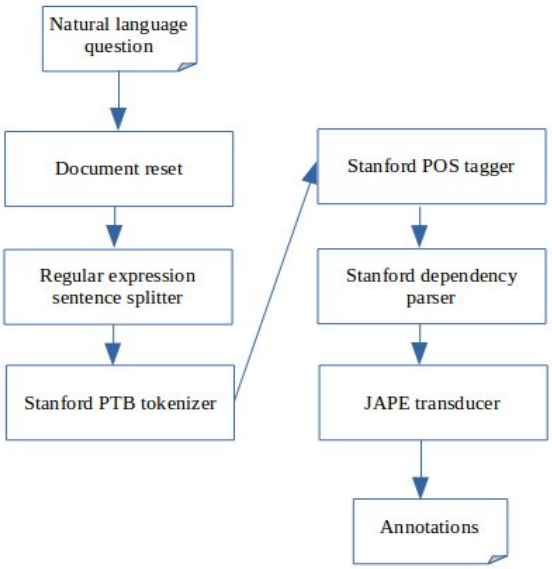

Figure 5.2: Processing pipeline

Consider question (1):

Question(1): “What is the total number of activities for each student in the course 1545”.

This sentence will pass through the pipeline process producing a set of annotations. The steps of this pipeline are (see figure 5.2):

- Document reset: clears all the annotations in the sentence.
- Regular expression sentence splitter: it splits the text into sentences.
- Stanford PTB tokenizer: it annotates tokens with the following information:
  - Type: a token

- Start: the index of the first character of the word.
- End: the index of the last character of the word.
- Id: the identifier of the token
- Features: some attributes of the word such as: kind, length, orth and string.

For example, the word “What” is tokenized as: type=Token, start=0, end= 4, Id=4, Features={kind=word, length=4, orth=upperInitial, string=What}.

- Stanford POS tagger: it will add the feature “category” to the token. For example, the token 4 will have the feature category=WP which means “Wh-pronoun”.
- Stanford dependency parser: it will add the feature “dependencies” to the token. For example, the token 13 will have the feature dependencies=[pobj(15)] which means that the token 15 is a prepositional object of the token 13.
- JAPE transducer: JAPE rules to extract the annotations from the GATE framework and store them in a file. JAPE (Java Annotation Patterns Engine) instructions are regular expressions on annotations.

Figure 5.3 presents the annotations generated for Question(1), in figure 5.4 we present a graphical representation of these annotations[1].

In the next section we present the algorithm to convert the annotations to a SPARQL query.

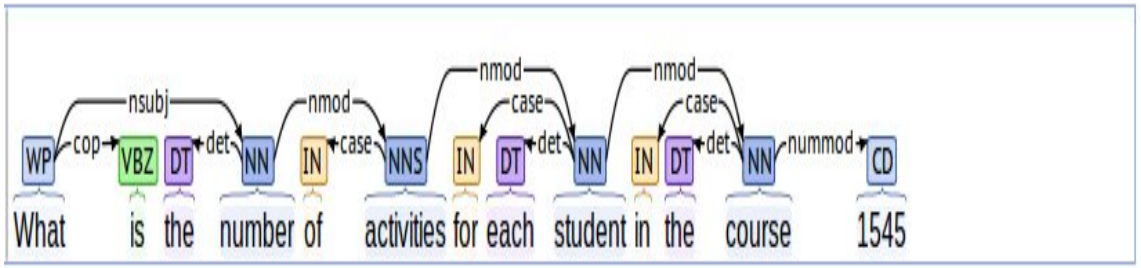

Figure 5.4: POS-tags and dependencies of Question(1)

5.2 Building SPARQL query

An algorithm was created to convert the generated annotations from the previous step to a SPARQL query. The algorithm is presented in figure 5.5.

[1]This graph was obtained using Stanford CoreNLP Online Demo

```
AnnotationImpl: id=4; type=Token;
features={string=What, category=WP, dependencies=[cop(5), nsubj(11)]}

AnnotationImpl: id=5; type=Token; features={string=is, category=VBZ}

AnnotationImpl: id=7; type=Token; features={string=the, category=DT}

AnnotationImpl: id=9; type=Token; features={string=total, category=JJ}

AnnotationImpl: id=11; type=Token;
features={string=number, category=NN, dependencies=[det(7), amod(9), prep(13), nmod(15)]}

AnnotationImpl: id=13; type=Token;
features={string=of, category=IN, dependencies=[pobj(15)]}

AnnotationImpl: id=15; type=Token;
features={string=activities, category=NNS, dependencies=[prep(17), nmod(21)]}

AnnotationImpl: id=17; type=Token;
features={string=for, category=IN, dependencies=[pobj(21)]}

AnnotationImpl: id=19; type=Token; features={string=each, category=DT}

AnnotationImpl: id=21; type=Token;
features={string=student, category=NN, dependencies=[det(19), prep(23), nmod(27)]}

AnnotationImpl: id=23; type=Token;
features={string=in, category=IN, dependencies=[pobj(27)]}

AnnotationImpl: id=25; type=Token; features={string=the, category=DT}

AnnotationImpl: id=27; type=Token;
features={string=course, category=NN, dependencies=[det(25), num(29)]}

AnnotationImpl: id=29; type=Token; features={string=1545, category=CD}
```

Figure 5.3: annotations for Question (1)

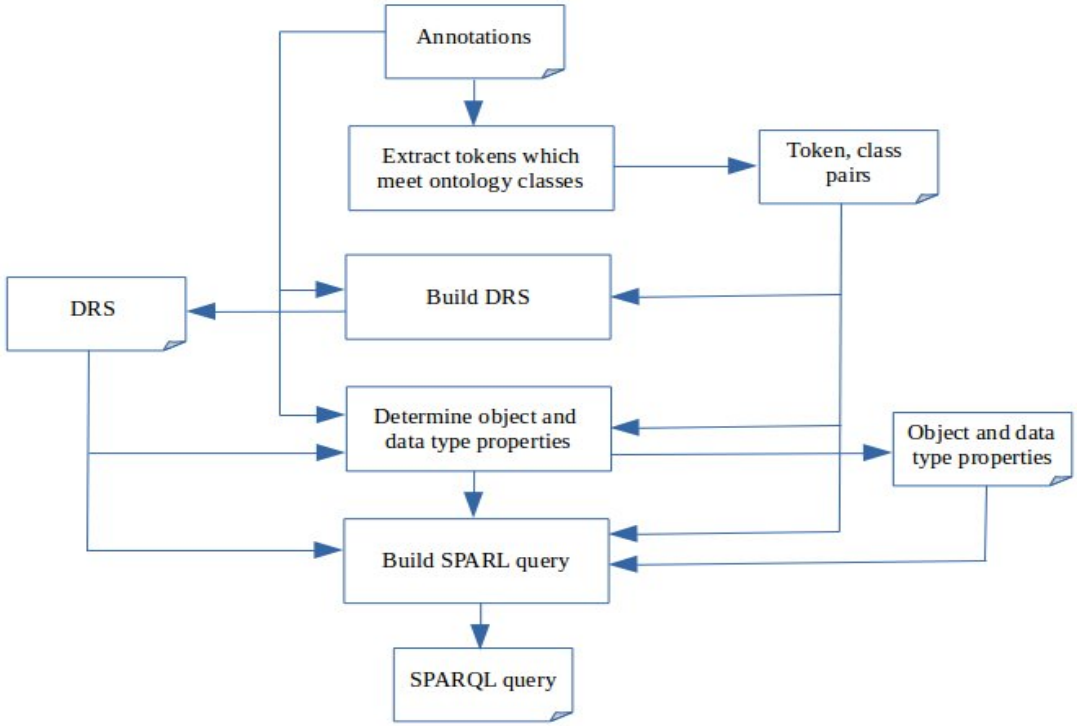

Figure 5.5: The SPARQL-query-builder algorithm

In the next subsections we explain the steps to interpret Question(1) as an example.

5.2.1 Discourse entities

Discourse entities are recognized in the process “Extract tokens which meet ontology classes”. The algorithm will check all the nouns in the sentence. All the tokens marked as “NN” or “NNS” are nouns. The discourse are candidates to match ontology classes. The tokens marked as “NNS” are in plural form so they were lemmatized to get the singular form. Then they are searched for in the ontology classes to find a match. For Question(1), the result is the following discourse entities:

- (A, 4, What, Wh).
- (X, 21, student, each)
- (Y, 27, course, the)
- (Z, 15, activities, -)
- (W, 11, number, the)
- (B, 29, 1545, -)

Then this module will try to match each discourse entity with ontology classes like following:

- The “student” token meets the “User” class: (X, 21, student, User)

- The “course” token meets the “Course” class: (Y, 27, course, Course)
- The “activities” token meets the “UserActivity” or the “CourseActivity” classes: (Z, 15, activities, [UserActivities, CourseActivities])
- the “number” token meets no class: (W, 11, number, null)

5.2.2 DRS conditions

The discourse representation conditions are obtained by extracting the dependencies of prepositional phrases, verbs and adjectives.

The algorithm loops through all the annotations to extract conditions. For each dependency in the tokens features it will get the head and dependent of it. The discourse conditions for Question(1) are:

- of(number-11, activities-15) -> of(W, Z)
- for(activities-15, student-21) -> for(Z, X)
- in(student-21, course-27) -> in(X, Y)
- is(What-4, number-11) -> is(A, W)
- total(number-11) -> total(W)
- num(course,1545) -> num(Y,B)

These conditions will be interpreted in the ontology in order to be translated into ontology properties.

5.2.3 Determine object and data type properties

The DRS will be used to determine the object and data type properties from the ontology.

- for(Z, X) will produce the “activityByUser” object property. It will be obtained by searching for the object properties between the two classes. This is achieved using Jena reasoner.
- in(X, Y) will produce the “isStudentOf” object property.
- of(W, Z) will produce the “num” data type property.
- num(Y, B) will produce the “c_id” data type property.

5.2.4 Generating SPARQL query

A SPARQL query is built using the DRS.

1. each discourse entity that was matched with a class will give rise to a variable definition in SPARQL query. (1), (2), (3) were generated by the tokens X, Y, Z respectively.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ns: <http://www.semanticweb.org/courses-ontology-57#>
 select ?x (sum(?num) as ?numAct)
  where {
         ?x rdf:type ns:User.              (1)
         ?y rdf:type ns:Course.            (2)
         ?z rdf:type ns:UserActivity.      (3)
         ?z ns:activityInCourse ?y.        (4)
         ?x ns:isStudentOf ?y.             (5)
         ?z ns:activityByUser ?x.          (6)
         ?z ns:num ?num.                   (7)
         ?y ns:c_id `1545'.                (8)
        } group by ?x
```

Figure 5.6: the activities of students in course '1545'

2. each object property or data type property will generate a constraint.

   (4), (5), (6) were generated using the object properties.

   (7) was generated using the data type property.

   (8) was generated using the num(Y, B) from DRS.

3. the select statement in SPARQL is built by collecting all the discourse variables with the "each" quantifier followed by variables that have "Wh" as a quantifier. So it was generated using total(W), each(X).

4. The variables which have the quantifier "each" will be presented in the group by clause. Group by was generated using each(X).

Figure 5.6 presents the SPARQL query generated as a result.

Figure 5.7 shows the result of the generated SPARQL query.

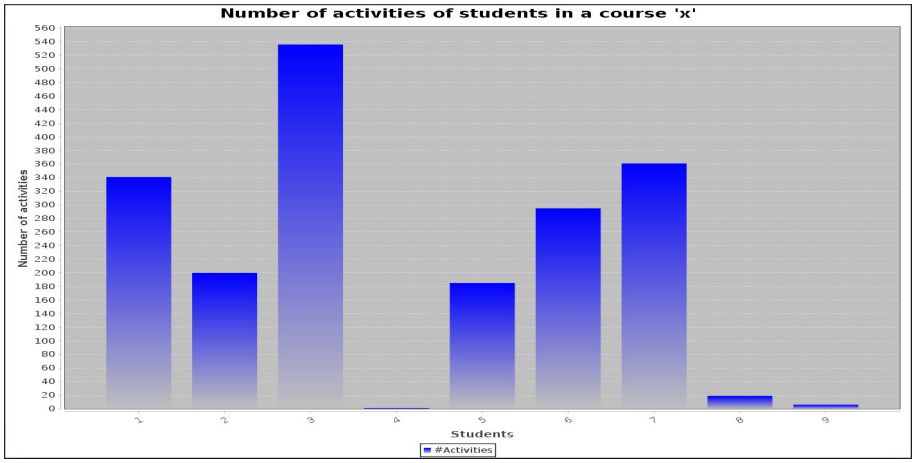

Figure 5.7: Students activities in course 'x'

6

Conclusion and Future Work

As presented in chapter 3, it was possible to achieve an information extraction system to extract information from the database of Moodle and populate it into an ontology.

As presented in chapter 4, it was possible to query the ontology to show statistics about the courses. This was not possible in the traditional Learning Management System.

As presented in chapter 5, it was possible to build a question answering system to answer natural language questions using Natural Language Processing o convert the natural language into a query language for the ontology (SPARQL).

As a future work:

- We can apply this work for another Learning Management System. Building ontologies for many Learning Management Systems make it possible to merge these ontologies into one comprehensive ontology in the domain.
- Natural Language Processing can be used to extract information from textual contexts of the courses resources. Achieving that make it possible to extract all the information of the Learning Management System and not only from the database of it.

- The question answering system can be improved to answer more questions with more accuracy. This can be achieved by improving the proposed algorithm which depends on Natural Language Processing.

Bibliography

[ACM+92] Alicia Ageno, Irene Castellón, Maria Antonia Marti, German Rigau, Francesc Ribas, Horacio Rodriguez, Mariona Taulé, and Felisa Verdejo. Seisd: an environment for extraction of semantic information from on-line dictionaries. In *Proceedings of the third conference on Applied natural language processing*, pages 253–254. Association for Computational Linguistics, 1992.

[AM02] Enrique Alfonseca and Suresh Manandhar. An unsupervised method for general named entity recognition and automated concept discovery. In *Proceedings of the 1st international conference on general WordNet, Mysore, India*, pages 34–43, 2002.

[APR12] Isabelle Augenstein, Sebastian Padó, and Sebastian Rudolph. Lodifier: Generating linked data from unstructured text. In *The Semantic Web: Research and Applications*, pages 210–224. Springer, 2012.

[AVH04] Grigoris Antoniou and Frank Van Harmelen. Web ontology language: Owl. In *Handbook on ontologies*, pages 67–92. Springer, 2004.

[BCRS06] Paul Buitelaar, Philipp Cimiano, Stefania Racioppa, and Melanie Siegel. Ontology-based information extraction with soba. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

[BK05] Nacéra Bennacer and Lobna Karoui. A framework for retrieving conceptual knowledge from web pages. In *SWAP*, pages 14–16. Citeseer, 2005.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. " O'Reilly Media, Inc.", 2009.

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[BOS03] P Buitelaar, D Olejnik, and M Sintek. Ontolt: A protege plug-in for ontology extractionfrom text. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2003.

[Bos08] Johan Bos. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 277–286. Association for Computational Linguistics, 2008.

[Bri99] Sergey Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183. Springer, 1999.

[CDF+00] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Seán Slattery. Learning to construct knowledge bases from the world wide web. *Artificial intelligence*, 118(1):69–113, 2000.

[CHS04] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM, 2004.

[CM14] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[CN02] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[Cun02] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[CV05] Philipp Cimiano and Johanna Völker. Towards large-scale, open-domain and ontology-based named entity classification. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*, 2005.

[DBVSW07] Viktor De Boer, Maarten Van Someren, and Bob J Wielinga. Relation instantiation for ontology population using the web. In *KI 2006: Advances in Artificial Intelligence*, pages 202–213. Springer, 2007.

[DEFS99] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. *Ontobroker: Ontology based access to distributed and semi-structured information*. Springer, 1999.

[DKP+04] Witold Drozdzynski, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. Shallow processing with unification and typed feature structures - foundations and applications. *KI*, 18(1):17, 2004.

[Emb04] David W Embley. Toward semantic understanding: an approach based on information extraction ontologies. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 3–12. Australian Computer Society, Inc., 2004.

[FG11] Carla Faria and Rosario Girardi. An information extraction process for semi-automatic ontology population. In *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*, pages 319–328. Springer, 2011.

[FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[FSE11] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[GMJ04] Avigdor Gal, Giovanni Modica, and Hasan Jamil. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, page 853. IEEE, 2004.

[GPMM03] A Gómez-Pérez and D Manzano-Mach. Deliverable 1.5: A survey of ontology learning methods and tools. *OntoWeb deliverable*, 2003.

[Gru93] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[H+02] Ian Horrocks et al. Daml+oil: A description logic for the semantic web. *IEEE Data Eng. Bull.*, 25(1):4–9, 2002.

[HC07] Brian Harrington and Stephen Clark. Asknet: Automated semantic knowledge network. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 889. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[Hea92] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[HEBR09] Maryam Hazman, Samhaa R El-Beltagy, and Ahmed Rafea. Ontology learning from domain specific web documents. *International Journal of Metadata, Semantics and Ontologies*, 4(1-2):24–33, 2009.

[HKR+04] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. *University of Manchester*, 2004.

[HR10] Jerry R Hobbs and Ellen Riloff. Information extraction. *Handbook of natural language processing*, 2, 2010.

[HSC02] Siegfried Handschuh, Steffen Staab, and Fabio Ciravegna. S-cream—semi-automatic creation of metadata. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 358–372. Springer, 2002.

[JM00] Dan Jurafsky and James H Martin. *Speech & language processing*. Pearson Education India, 2000.

[KAB04] Lobna Karoui, Marie-Aude Aufaure, and Nacera Bennacer. Ontology discovery from web pages: Application to tourism. In *In the Workshop of Knowledge Discovery and Ontologies*. Citeseer, 2004.

[Kam81] Hans Kamp. A theory of truth and semantic representation. *Formal semantics: The essential readings*, pages 189–213, 1981.

[KC06] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. 2006.

[KJ15] Deepika Kumawat and Vinesh Jain. Pos tagging approaches: A comparison. *International Journal of Computer Applications*, 118(6), 2015.

[KM02] Taku Kudo and Yuji Matsumoto. Japanese dependency analysis using cascaded chunking. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[KM03] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[LMP01] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[MC06] Luke K McDowell and Michael Cafarella. *Ontology-driven information extraction with ontosyphon*. Springer, 2006.

[Mic13] Joel Mickelin. Named entity recognition with support vector machines, 2013.

[MMS93] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[MNS03] Alexander Maedche, Günter Neumann, and Steffen Staab. Bootstrapping an ontology-based information extraction system. In *Intelligent exploration of the web*, pages 345–359. Springer, 2003.

[MPRH05] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[MS00] Alexander Maedche and Steffen Staab. The text-to-onto ontology learning environment. In *Software Demonstration at ICCS-2000-Eight International Conference on Conceptual Structures*, volume 38, 2000.

[MSB+14] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[NFM00] Natalya Fridman Noy, Ray W Fergerson, and Mark A Musen. The knowledge model of protege-2000: Combining interoperability and flexibility. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 17–32. Springer, 2000.

[OAH+07] Daniel Oberle, Anupriya Ankolekar, Pascal Hitzler, Philipp Cimiano, Michael Sintek, Malte Kiesel, Babak Mougouie, Stephan Baumann, Shankar Vembu, Massimo Romanelli, et al. Dolce ergo sumo: On foundational and domain models in the smartweb integrated ontology (swinto). *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(3):156–174, 2007.

[PDG12] Valentina Presutti, Francesco Draicchio, and Aldo Gangemi. Knowledge extraction based on discourse representation theory and linguistic frames. In *Knowledge Engineering and Knowledge Management*, pages 114–129. Springer, 2012.

[PKO+04] Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov. Kim-a semantic platform for information extraction and retrieval. *Natural language engineering*, 10(3-4):375–392, 2004.

[Pra04] K Prantner. Ontour: The ontology.[online:] http://e-tourism. deri. at/ont/docu2004, 2004.

[RJ+99] Ellen Riloff, Rosie Jones, et al. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[RMMGCN+11] Juana Marıa Ruiz-Martınez, Jose Antonio Minarro-Giménez, Dagoberto Castellanos-Nieves, Francisco Garcıa-Sánchez, and R Valencia-Garcia. Ontology population: an application for the e-tourism domain. *International Journal of Innovative Computing, Information and Control (IJICIC)*, 7(11):6115–6134, 2011.

[Sar08] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008.

[SBB+99] Steffen Staab, Christian Braun, Ilvio Bruder, Antje Düsterhöft, Andreas Heuer, Meike Klettke, Günter Neumann, Bernd Prager, Jan Pretzel, Hans-Peter Schnurr, et al. Getess—searching the web exploiting german texts. In *Cooperative Information Agents III*, pages 113–124. Springer, 1999.

[SDGDG11] Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane, and Brigitte Grau. An ontology for the conceptualization of an intelligent environment and its operation. In *Artificial Intelligence (MICAI), 2011 10th Mexican International Conference on*, pages 16–22. IEEE, 2011.

[SDGDG13] Driss Sadoun, Clemence Dubois, Yacine Ghamri-Doudane, and Brigitte Grau. From natural language requirements to formal specification using an ontology. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 755–760. IEEE, 2013.

[Set04] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 104–107. Association for Computational Linguistics, 2004.

[SFK06] György Szarvas, Richárd Farkas, and András Kocsor. A multilingual named entity recognition system using boosting and c4. 5 decision tree learning algorithms. In *International Conference on Discovery Science*, pages 267–278. Springer, 2006.

[SKW08] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[SM01] Gerd Stumme and Alexander Maedche. Fca-merge: Bottom-up merging of ontologies. In *IJCAI*, volume 1, pages 225–230, 2001.

[SS04] Yusuke Shinyama and Satoshi Sekine. Named entity discovery using comparable news articles. In *Proceedings of the 20th international conference on Computational Linguistics*, page 848. Association for Computational Linguistics, 2004.

[ST08] Kenji Sagae and Jun'ichi Tsujii. Shift-reduce dependency dag parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 753–760. Association for Computational Linguistics, 2008.

[SWGM05] Marta Sabou, Chris Wroe, Carole Goble, and Gilad Mishne. Learning domain ontologies for web service descriptions: an experiment in bioinformatics. In *Proceedings of the 14th international conference on World Wide Web*, pages 190–198. ACM, 2005.

[TKMS03] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the*

North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, pages 173–180. Association for Computational Linguistics, 2003.

[TL12] Theerayut Thongkrau and Pattarachai Lalitrojwong. Ontopop: An ontology population system for the semantic web. *IEICE TRANSACTIONS on Information and Systems*, 95(4):921–931, 2012.

[TM00] Kristina Toutanova and Christopher D Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000.

[VNCN05] Paola Velardi, Roberto Navigli, Alessandro Cuchiarelli, and R Neri. Evaluation of ontolearn, a methodology for automatic learning of domain ontologies. *Ontology learning from text: Methods, applications and evaluation*, 123:92–106, 2005.

[WD10] Daya C Wimalasuriya and Dejing Dou. Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Science*, 2010.

[WHW08] Fei Wu, Raphael Hoffmann, and Daniel S Weld. Information extraction from wikipedia: Moving down the long tail. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 731–739. ACM, 2008.

[WHWHG04] Peter Wiemer-Hastings, K Wiemer-Hastings, and A Graesser. Latent semantic analysis. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pages 1–14. Citeseer, 2004.

[WKR10] René Witte, Ninus Khamis, and Juergen Rilling. Flexible ontology population from text: The owlexporter. In *LREC*, volume 2010, pages 3845–3850, 2010.

[WLB12] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. *ACM Computing Surveys (CSUR)*, 44(4):20, 2012.

[WW07a] William R Watson and Sunnie Lee Watson. What are learning management systems, what are they not, and what should they become. *TechTrends*, 51(2):29, 2007.

[WW07b] Fei Wu and Daniel S Weld. Autonomously semantifying wikipedia. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 41–50. ACM, 2007.

[YM07] Burcu Yildiz and Silvia Miksch. ontox-a method for ontology-driven information extraction. In *Computational Science and Its Applications–ICCSA 2007*, pages 660–673. Springer, 2007.

[ZS02] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger. In *proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 473–480. Association for Computational Linguistics, 2002.