Bachelor Thesis Economics & Informatics

Semi-Automatic Information Extraction Using Ontologies

Author:
Wouter IJntema
wouterijntema@student.eur.nl
306742wi

Supervisor:
Dr. Flavius Frasincar
frasincar@ese.eur.nl

Co-Reader:
Frederik Hogenboom, MSc
fhogenboom@ese.eur.nl

Erasmus School of Economics
Erasmus University Rotterdam
July 13, 2009

Abstract

The vision of the Semantic Web is to structure the World Wide Web in such a way that it is not only understandable for humans, but also for computers. At the heart of the Semantic Web lay the ontologies, that describe the meaning of concepts in a certain domain. These are mostly created and maintained by domain experts.

In this Bachelor Thesis we propose a rule based ontology learning process to help domain experts with this process. First we discuss the exiting approaches in ontology learning and pattern matching. Inspired by those approaches, we describe a syntax to describe information extraction rules in order to update the ontology. We show that employing ontologies in the rules, helps them to be created in a more generic manner. This way, different kinds of information may be extracted with the same rule.

Acknowledgements

First of all, I would like to thank my supervisor, Flavius Frasincar for helping and encouraging me to write this thesis. His knowledge and inspiration really helped me with the process. Secondly I would like to thank Frederik Hogenboom for being my co-reader. Last but not least, I would like to show my gratitude to Marion and my parents, for all their support during the writing of my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is a vast collection of unstructured information. For human beings it is readable and understandable, but for computers it is only readable. With the Semantic Web [3] the World Wide Web Consortium (W3C) provides a framework to add structure to the information. One of the formats to represent this structure is the Web Ontology Language (OWL) [2].

Ontologies can be used to store domain specific knowledge in the form of concepts with all kinds of relations between the concepts. We denote these relationships with triples that consist of an object, predicate and a subject. For instance the competitor relationship between two companies can be described with 'CompanyX competitor-of CompanyY'. With an ontology that is well-structured, it is possible to search for specific relations. For example the question 'Who are the competitors of Microsoft?' can easily be answered.

[14] proposes a personalized news service based on Semantic Web technologies. At the heart of this news service is a knowledge base in the form of an ontology that contains concepts with all sorts of relations between them. By linking the concepts from the news items to the concepts in the knowledge base it provides us with the possibility to retrieve news items related to specific concepts. For instance you are able to get all the news from Microsoft and its competitors. This saves an enormous amount of time when compared to traditional text-based searching. Besides searching for news items the ontology can also be used for recommending news items based on the users previous read items.

Creating and maintaining a knowledge base requires a domain expert to invest a lot of time. Certain events in the news will cause relations to stop being valid. For instance, if Microsoft buys Google, it means that the competitor relation between these companies no longer exists. In this bachelor thesis we will focus on a (semi)automatic way of information extraction from financial news with the purpose to decrease the time needed to maintain a knowledge base.

1.1 Research Goal

The main problem with domain ontologies is that it requires expert knowledge and it is a time consuming job when building them manually. Therefore methods for extracting knowledge from text in an automatic way are required. Prior approaches have been focussing on lexico-syntactic patterns for information extraction, which leaves room for ambiguity. In our approach we will employ ontologies to create lexico-semantic patterns.

1.1.1 Research Question

In this thesis we will focus on pattern-based information extraction from financial news items. To achieve the research goal, we will answer the following research question:

> *To what extent can we automatically discover and extract financial information in emerging news by using ontologies?*

In order to answer this question we will answer the following sub-questions:

1. How can we use patterns for information extraction?
2. What grammar can be used to describe information extraction rules?
3. How can we extend this grammar to employ ontologies in the information extraction process?

1.2 Methodology

To be able to answer the research questions we use a few different methodologies. First we start with a literature review to evaluate the existing technologies. Second, we construct a syntax for the rules that can be used to extract information from text. This syntax is inspired by work that is discussed in the literature review. After the rule syntax is formulated, an implementation of the proposed framework is presented. To construct this implementation some existing Java libraries will be used, like GATE [9] and Jena [22]. The implementation will be an extension to the Hermes News Portal [14], which is a personalized news service based on Semantic Web techniques. To evaluate the ontology learning process based on information extraction rules, we present two use cases that discuss the strengths and weaknesses of the approach.

1.3 Structure

The structure of this thesis is as follows.

In chapter 2 we focus on the related work, where we first discuss natural language processing in general is discussed, followed by the ontology learning process. Then we focus on pattern matching and the existing approaches. We conclude this chapter with two examples of ontology learning projects.

Chapter 3 discusses the formulation of the rule syntax and the features that can be used in the construction of the rules. We also discuss how ontologies can be employed to add semantics to the rules.

After the formulation of the rule syntax, chapter 4 discusses the implementation that employs the information extraction rules to extract information from news items.

The evaluation in chapter 5 presents two extensive use cases to highlight the strengths and weaknesses of the proposed approach.

Chapter 6 concludes this thesis with the conclusions where we give an answer to the formulated research question. In the end we identify future research directions.

Chapter 2

Related Work

When dealing with a knowledge based system it is important to construct and maintain a good information base. This chapter discusses existing techniques in the (semi)automatic information extraction from text. First in section 2.1, we will present a short introduction to the field of Natural Language Processing (NLP). Section 2.2 will discuss ontology learning, followed by section 2.3 which will tackle the pattern matching approach for information extraction. We conclude this chapter with section 2.4 which will show two examples of information extraction systems.

2.1 Natural Language Processing

Alan Mathison Turing proposed in [24] that if a human judge engages in a natural language conversation with one human and one machine, each of which tries to appear human, and the judge could not distinguish the human from the computer, the computer has passed the intelligence test. A small part of the intelligence comes from the processing of the natural language.

We distinguish two forms of natural language processing. First, natural language generation concentrates on creating human readable text from well-structured computer databases. Second, natural language understanding which has as goal to convert human readable text into formal representations that are understandable for the computer.

For our research, we will focus on the understanding of human languages. By 'understanding' we mean, knowing what a word represents in the current context and how to relate it to other concepts in a meaningful way.

In the field of natural language processing, there are numerous software packages available. We have chosen the General Architecture for Text Engineering (GATE) [9] for text processing, because it is widely used and the pipeline architecture enables new processing components to be easily plugged in.

2.1.1 General Architecture for Text Engineering (GATE)

The General Architecture for Text Engineering is a text annotation framework which provides both a graphical development environment and a Software Development Kit (SDK). GATE can be used for the development and deployment of software applications that employ natural language processing techniques. One of the advantages of GATE is its modular structure. This enables developers and researcher, besides using out-of-the-box components, to create their own components and adapt the system to their needs.

Additionally, GATE is employed with several information extraction tools, one of which is a Java Annotation Patterns Engine (JAPE) [10]. JAPE provides finite state transduction over annotations based on regular expressions. In our research we will implement a rule engine that processes the rules in a similar manner.

2.2 Ontology Learning

A great deal of the technologies that employ the Semantic Web, as proposed by [3], rely on ontologies. As [15] and [16] describe, the term *ontology* comes from the field of philosophy that is concerned with the study of being or existence. In the 1980's researchers in the Artificial Intelligence field have adopted the term to refer to both a theory of a modeled world and a component of knowledge systems. The W3C proposes a Semantic Web standard, OWL [2], to formally describe ontologies as *specifications of conceptualizations.*

The ontology consists of two layers, the Terminological-Box (TBox) and the Assertion-Box (ABox) [1]. The TBox is the description of a domain in the form of a terminology, for instance a set of classes and properties. The ABox contains the actual knowledge about the individuals in the domain.

In *ontology learning* we deal with the acquisition of knowledge, and in this context, particularly the acquisition of knowledge from text. Ontology learning distinguishes itself from general knowledge acquisition in three ways. First, because of the Semantic Web that is generally involved with ontology learning, it attracts researches from a very broad variety of disciplines: knowledge representation, logic, philosophy, databases, machine learning and natural language processing. Second, ontology learning commonly focuses on the extraction from and for Web content instead of homogeneous data collections. Third, the systematic evaluation methods that are used in machine learning are now being applied to ontology learning. Due to those well-defined evaluation methods, only competitive and demonstrable approaches survive. [6]

The ontology learning layer cake [6], shown in figure 2.1, distinguishes six tasks in ontology learning. In our approach we will use semantic-based rules

to identify concepts and relations between concepts in news items, which will cover the first five layers (from bottom to top) of the ontology learning cake.

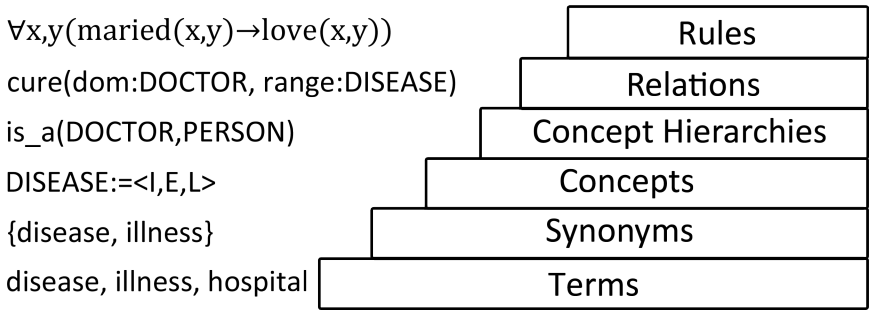

Figure 2.1: Ontology Learning Layer Cake

To cope with ontologies we use the Jena [22] package, which provides a Semantic Web framework for Java.

2.3 Pattern Matching

Ontology learning can be done in several ways. For instance, concept induction, which tries to define a hypothesis about a relation, that is consistent with examples. [20] proposes concept induction by using a term subsumption formalism and a learning algorithm to construct a knowledge base. [21] gives a comparison of the different methods that can be used to create ontologies from text, for instance clustering, association rules, classification and conception mining. In our approach we will focus on pattern matching. The major advantage of pattern matching above statistical methods is that pattern matching only requires one expressed instance of a relation while others require a statistically significant number of expressed relations [17].

[17, 18] proposes the use of lexico-syntactic pattern based information extraction. This approach tries to find hyponym and hypernym relations by discovering regular expression patterns in free text. An example is the application of pattern 2.1 to sentence 2.2.

$$\textit{such NP as } \{\textit{NP,}\}^* \{\textit{(or—and)}\} \textit{ NP} \tag{2.1}$$

... works by such authors as Herrick, Goldsmith, and Shakespeare. (2.2)

This results in the following found relationships:

$$\begin{aligned} &\textit{hyponym(“author”, “Herrick”)} \\ &\textit{hyponym(“author”, “Goldsmith”)} \\ &\textit{hyponym(“author”, “Shakespeare”)} \end{aligned} \tag{2.3}$$

Such patterns do not capture the context of the text. Our intent is to explore the possibilities of adding semantics to the patterns by using Semantic Web technologies. The main advantage of lexico-semantic patterns are that such patterns take advantage of the corpus data which helps the parser cope with the complexity and flexibility of real text [19].

2.3.1 ANNIE with JAPE Rules

GATE is being distributed with an Information Extraction (IE) system, called ANNIE (A Nearly-New Information Extraction System) [11]. For information extraction ANNIE uses annotations. An annotation is characterized by a *type* and a set of *features*. The annotation sets store the annotation in a structured manner and constitute independent layers of an annotation over the text content. The generality of annotation sets enable them to represent a wide variety of linguistic information. ANNIE relies on JAPE, a Java Annotation Patterns Engine [10], to provide a rule-based pattern engine to process annotated text. In JAPE rules there are a left-hand-side (LHS) part and a right-hand-side (RHS) part. The left-hand-side consists of an annotation pattern that may contain regular operators (e.g., ‘*’, ‘?’, ‘+’). The right-hand-side consists of annotation manipulation patterns.

There are three main modules in ANNIE. First the tokenizer, which splits the text into simple tokens such as words, numbers, and punctuation. Second the gazetteer lookup module, which is used to identify special key words, like countries, currencies, or organizations. The third module is the JAPE grammar. The following example (obtained from [10]) shows a rule for recognizing an IP-address[1] inside angled brackets:

```
Rule: IPAddress
({Token.kind == number}
 {Token.string == "."}
 {Token.kind == number}
 {Token.string == "."}
 {Token.kind == number}
 {Token.string == "."}
 {Token.kind == number}
)
:ipAddress -->
 :ipAddress.Address = {kind = "ipAddress"}
```

(Rule 1)

[1]This rule ignores the range of the numbers.

The example consists of five parts. First the definition of the rule, which is called 'IPAddress' in this case. Second the rule itself, which is enclosed by the rounded braces. This rule matches four consecutive occurrences of a token of the kind 'number', followed by a dot. Third, the rule is assigned a label, which is always denoted by a preceding colon. Fourth, information about the annotation is transferred from the LHS of the rule using the label just described, and annotated with the entity type (which follows it). Finally, attributes and their corresponding values are added to the annotation [11].

ANNIE focuses on lexical-syntactic analysis, which is fairly limited. As the lexical-syntactic patterns do not take into account the semantic context of the text, which leaves room for ambiguity, in our approach we aim at exploiting the lexico-semantic patterns to overcome this issue.

2.3.2 CAFETIERE

CAFETIERE (Conceptual Annotations for Facts, Events, Terms, Individual Entities, and RElations) [4] is another rule-based system for ontology driven temporal text mining. First CAFETIERE applies several preprocessing techniques to the text, i.e., tokenization, part-of-speech tagging, and gazetteer lookup. To extract information from text, a rule notation is defined. A rule has the following form:

```
A => B \ C / D                                                    (Rule 2)
```

where A represents the phrase that is recognized, B (optional) represents the text prior to C, C defines the text elements that are part of the phrase and D (optional) is the neighboring text immediately following C.

A simple example of a rule that would match an expression like '40 mg' is:

```
[syn=NP, sem=QTY] => \[syn=CD], [sem=measure]/;                   (Rule 3)
```

In this rule *syn* and *sem* indicate the syntactic and respectively semantic category, which are here NP (noun phrase) and QTY (quantity) on the left hand side, while on the right hand side requires the syntactic category of CD (cardinal digit) followed by the semantic category *measure*. If the right hand side matches, the text will be annotated as described by the left hand side. This rule is context free, since it has no text units to the left of \ and none to the right of /. CAFETIERE also takes into account the ordering of the rules. When one rule matches the text and annotates the text, the original annotation might no longer be visible to the next rule. For instance consider the following tokens with the corresponding syntactic and semantic category:

```
Token:  Junior  programmer
  syn:  JJ      NN
  sem:          job
```

Suppose we have the following two rules:

```
[syn=NN] => \[syn=JJ|NN]{1,3}, [syn=NN]/;                    (Rule 4)
```

```
[syn=NN, sem=job] => \[syn=JJ|NN]{1,3}, [sem=job]/;          (Rule 5)
```

Rule 4 will match the text "Junior" and "programmer" to rewrite the token sequence as

```
Token:  Junior  programmer
  syn:  NN
  sem:
```

Since "Junior programmer" is now annotated as one single text unit, rule 5 can no longer succeed. This also holds for when rule 5 is being applied before rule 4.

In our work we will benefit from the work that is done in the CAFETIERE project, e.g., the rule notation. The limitation for CAFETIERE is the same as for ANNIE. As the rules are defined on lexico-syntactic level the approach fails to include domain semantics. Both the gazetteer and the lexico-semantic rules might benefit from an ontology-based approach.

2.4 Example Projects

2.4.1 KnowItAll

KnowItAll [12] is a system that aims at the automatic extraction of knowledge from the web. By using information extraction rules, similar to the rules described in [17, 18], a search engine and statistics they managed to extract 54.753 facts in four days. Different from our approach is that they use text gathered by a search engine while we use news items coming from RSS feeds, to update the ontology. Additionally, we make use of Semantic Web techniques, while their approach uses traditional relational database to store the extracted relations.

2.4.2 PANKOW

Another project that tries to overcome the tedious work that comes with the annotation of text, is PANKOW [7, 8]. As we try to exploit the lexico-semantic patterns to extract information from news items, we differentiate from PANKOW, because in their approach lexico-syntactic patterns are used to annotate text found on the Web by using *Google*.

Chapter 3

Rule Syntax

To be able to extract information from text we will be using rules. With these rules patterns in the text can be described. These patterns can contain new concepts that are not yet known in the knowledge base. Besides discovering new concepts, rules can also be used for the discovery of events, i.e., a company introduces a new product will add a new relation between the product and the company. This chapter focuses on the formulation of the rule syntax. In section 3.1 we will describe the syntax of the patterns that will be used. After that, section 3.2 will describe how to describe lexico-semantic patterns by employing ontologies in the rules. Finally we will give some concluding remarks in section 3.3.

3.1 Patterns

The texts are mined for the occurrence of the pattern that is defined. Defining such a pattern can be done in two different ways. The first definition is intuitive and is based on the approach proposed by Hearst [18]. The second definition is inspired by CAFETIERE [4] and is somewhat more extended.

3.1.1 Basic Syntax

With the description of the basic syntax we will explain how one can employ ontologies in the description of information extraction rules.

The sentence "*Google introduces its new browser Chrome*", contains three main parts. The first is *Google*, which is the subject. The second is *introduces*, which defines the relation and the third is the object of the relation, *Chrome*

The ontology consists of classes, such as companies, relations and products. If we assume that *Google* is in the ontology known as an instance of the class *Company* and *Introducing* is an instance of the class *Relation*. With Rule 6 we can annotate this sentence. In this rule, the prefix `cls` refers to

a class in the ontology and the prefeix `inst` refers to an instance of a class (denoted by the part after the prefix, i.e., `Introducing`) in the underlying ontology. `NNP` stands for Proper Noun, which is a noun that starts with a capital letter.

```
cls:Company inst:Introducing NNP
```
(Rule 6)

If this rule matches a sentence, one can decide to annotate the token that is annotated with NNP with the class *Product*. This requires additional features added to the rule syntax. In the next section we will discuss how we can define a syntax that is able to annotate the found tokens.

3.1.2 Extended Syntax

In this section we will focus on the definition of a new rule syntax that is inspired by the syntax proposed by [4] and the programming language *Prolog*. In Prolog clauses are defined as follows:

```
Head :- Body
```
(Rule 7)

where head is true if body is true. Based on this we can define the syntax of the grammar. Each rule has the following form:

```
A :- B \ C / D
```
(Rule 8)

where A is the phrase that is recognized and will be annotated as such if the right hand side is true. C represents the text elements that are part of the phrase. B, which must be present immediately prior to C and D, which must be present immediately following C, can used to add context to the rule. Both, B and D are optional. A context free rule can be described as follows:

```
A :- \ C /
```
(Rule 9)

Note that we have replaced the `"=>"`, that was proposed in [4], with `":-"`, because we found `"=>"` confusing, since it is generally used as an implication sign.

3.1.3 Regular Expression Operators

To be able to determine that certain parts of the expression are repetitive, standard regular expression operators are used. First, to indicate that the preceding element occurs *zero or one time*, we use the question mark (?).

Second, for *zero or more* times an asterisk (*) should be used. Finally, to indicate that the preceding element occurs *one or more times*, we use a plus sign (+).

In addition to the unlimited operators it is also possible to add a limit to the upper and lower boundaries. These limits should be put between braces, e.g., {2,4} means that the preceding element must occur at least twice and may occur up to four times.

3.1.4 Features

To be able to create strong and generic rules we propose to use the following features: token, syntactic category and orthography. The semantic category will be covered in section 3.2, since the lexico-semantic patterns will be employed by using an ontology.

For each feature-value combination a comparison operator is required. The operators that are allowed are `=` (equal) and `!=` (unequal).

Token

In some rules it is often useful to employ plain text, this can be done by using the `token` feature. In plain text you can define anything. An example of using the `token` feature is:

```
[syn=NG] :-
 [token="products by Google are:"]                    (Rule 10)
 \[syn=NN|JJ]*, [token=","]*/;
```

This rule matches a list with products manufactured by *Google*. The `token` feature is used twice. First to indicate the context ("products by Google are:") and second to match a comma (","). The other feature, is `syn` will be discussed in the next section.

The `token` feature might also contain *wildcard characters*. The asterisk sign (*) is used to match zero or more characters and the question mark (?) matches a single character.

Syntactic Category

Another feature that can be used in the description of a pattern, is the syntactic category of a word, denoted by `syn`. Examples of possible values of the syntactic category are shown in table 3.1.
In addition to the predefined part-of-speech tags, it is also possible to define custom values. In Rule 10 for example, `NG` denotes a Noun Group.

Abbreviation	Meaning
CC	Coordinating conjunction
CD	Cardinal number
IN	Preposition
JJ	Adjective
NN	Noun
NNP	Proper Noun
PP	Pronoun
RB	Adverb
UH	Interjection
VB	Verb, base form
VBZ	Verb, 3rd person singular present

Table 3.1: Basic Part-Of-Speech Tags

Orthography

A word is defined by a set of upper and lower case letters. A feature that can be employed to denote the orthography of a word is `orth`. There are four values defined: `upperInitial`, `allCaps`, `lowerCase`, `mixedCaps`. An example of the use of this feature is when annotating a list of names, `orth` may be set to the value `upperInitial`.

Labels

With labels, it is possible identify specific tokens in the text. For example, when used on the right-hand side of a rule it can be referred to on the left-hand side. This feature can be used in the following manner:

```
[label=labelname]
```
(Rule 11)

3.2 Ontologies Employed in Rules

By employing ontologies in the rules it becomes possible to add semantics to the rules. For instance if there is a news article about *Google* introducing a new product, like *Chrome*, and *Google* exists in the knowledge base it is possible to annotate the lexical representation of *Chrome* as a product and add a product-relation between *Chrome* and *Google*. In our implementation we make use of OWL to describe ontologies. When ontologies are employed in the rules it is possible to use one rule to describe multiple lexical representations. In this example three features of an OWL ontology occur. First, company is a *class*. Second, *Google* and the product are an *instance* of a

class, and third the relationship between Google and the product, which is denoted as an *object property*. In this section we will discuss how those three features of the ontology can be employed in information extraction rules.

3.2.1 Classes

Classes are a group of individuals that belong together because they share the same properties [2]. For example, *Google* and *Microsoft* both belong to the same class: *Company*. Other examples of classes are: *Product*, *People* and *Country*. In information extraction it is useful to look for specific classes in the text, such that it is possible to annotate new instances of classes. A class in a rule is denoted by the feature `class`.

For example, the following headline: *Google introduces Chrome*, can be annotated by:

```
[class=Product] :-
 [class=Company], [token="introduces"]
 \[syn=NN, orth=upperInitial]/
```
(Rule 12)

Assuming that *Google* already exists in the ontology, this rule will annotate *Chrome* as being of the class *Product*. Note that the relation between *Google* and the *Chrome* is not yet recognized, this is an object property and will be discussed in section 3.2.3.

Class Reference

With the `classOf` feature it is possible to annotate a token with the same class as another token that precedes or follows it in the context. The value of `classOf` feature needs to be same as the `label` feature of the token it refers to. In the first use case in section 5.2 we show how this feature can be used.

Subclasses

The class structure is a hierarchy. For instance, *Company* is a superclass of *NasdaqCompany*. If we assume that *Google* belongs to the class *NasdaqCompany*, we can infer in rule 12, that *Google* also belongs to the class *Company*. The other way around is of course, not per definition true.

3.2.2 Instances

An object created by a class is an instance of the class. For example, *Google* and *Microsoft* are instances of the class *Company*. Instances can be used to be more specific and are denoted by the feature `inst`. Instances are generally used on the right-hand side of the rule.

Rule 12 is an example of a triple pattern, which consists of a *subject* (Google), an *object* (Product) and a *relation* (introduces) between the subject and the object. While "Relation" is also a class in the ontology, it is possible to describe multiple lexical representations of the relation "introduces" with the same pattern. Rule 13 will also match the headline *Microsoft presents Windows*.

```
[class=Product] :-
 [class=Company], [inst=Introduce, class=Relation]
 \[syn=NN, orth=upperInitial]/
```
(Rule 13)

3.2.3 Object Properties

An object property is a property for which the value is an individual (or instance). Employing object properties in rule makes it possible to add specific relations between two annotated instances. For example the product-relation between *Google* and *Chrome*. In an information extraction rule we denote the object property feature as follows:

```
[(inst=subject|instref=subjectref),
 prop=propertyname,
 class=objectclass]
```
(Rule 14)

To create a link from the property to the subject, there are two options. The first is to specify the subject manually by using the `inst=subject` feature. The second option is to use the `instref=subjectref` to refer to a labeled token in the rule. Besides the subject, it is also required to specify the property name and the class of the object.

To continue the previous example, we can now add the relationship between the product and the company. The following rule will match the sentence *Microsoft presents Windows* and will annotate *Windows* as being a product of *Microsoft*.

```
[instref=comp, prop=productBy, class=Product] :-
 [label=comp, class=Company],
 [inst=Introduce, class=Relation]
 \[syn=NN, orth=upperInitial]/
```
(Rule 15)

3.2.4 Datatype Properties

As object properties link individuals to individuals, datatype properties link individuals to data values. An example of a datatype property of the class *Company* can be the year in which it was founded.

In the rules, the datatype property is defined by:

```
[(inst=subject|instref=subjectref),
 dataprop=propertyname,
 valueref=valueref]
```
(Rule 16)

where the instance is again either defined by a specific instance (by using `inst`) or by the reference to a instance found in the right-hand side of the rule (by using `instref`). The property name can refer to any datatype property that is defined in the ontology. The value of the data property is denoted by the `valueref` feature, which refers to a token on the right-hand side of the rule.

For example the simple sentence *"Apple is founded in 1976"* can be matched by the following rule:

```
[instref=comp,
 dataprop=founded,
 valueref=year] :-
 [class=Company label=comp],
 [class=Relation, inst=founding]
 \[syn=CD, label=year]/
```
(Rule 17)

3.3 Conclusion

In this section we have described the rule syntax for extracting information from news items. We first discussed the basics of syntax and the features that can be used. Followed by a method to employ ontologies in the rules, which results in powerful lexico-semantic rules. The advantage of such rules is that with the annotation of news items it is possible to look for classes in the text, discover new instances of specific classes, create relationships between certain instances by using object properties and add datatype properties to instances.

Chapter 4

Implementation

In this chapter we will discuss the implementation of the rule syntax. The information extraction component will be an extension to the Hermes News Portal [14], which is a framework for personalizing news, using Semantic Web techniques. The Hermes Framework and its implementation will be discussed in section 4.1. In section 4.2 we discuss what is done in the GATE pipeline before the rules will be processed. The rule engine will be discussed in section 4.3. Section 4.4 presents the the rule editor, followed by the validation procedure in section 4.5. In section 4.6 we will give some concluding remarks.

4.1 Hermes

4.1.1 Hermes Framework

The Hermes framework [14] is a proposition of steps to be followed in order to build a personalized news service. The system can be described by input, consisting of RSS feeds from various sources and output, news items fulfilling the user needs. At the heart of this personalized news service is a domain ontology, which is used to store information about the news items. The ontology is developed by domain experts, starting with the most salient concepts and then refining them using generalization/specification.

In addition to manually maintaining the ontology, Hermes also provides a series of news classification steps that help in extending the ontology. Techniques that are used are: tokenization, sentence splitting, part-of-speech tagging, morphological analysis and word sense disambiguation.

4.1.2 Hermes News Portal

The implementation of the Hermes framework is the The Hermes News Portal (HNP), which allows users to formulate queries and execute them on the domain ontology in order to retrieve relevant news items.

For the implementation various Semantic Web techniques are employed. The domain ontology is represented in OWL [2], querying is done with SPARQL [23], and time functionalities were added to SPARQL, which results in tSPARQL [14]. The classification of the news articles is done using GATE [9] and the WordNet [13] semantic lexicon. The classification occurs prior to our rules that will actually extract the information.

4.2 Preprocessing

Before the rules can be employed to match patterns in text, a few processing tasks need to be performed, like tokenisation, sentence splitting, part-of-speech tagging, morphological analysis and word sense disambiguation, needs to be done first.

4.2.1 GATE

This preprocessing is done using GATE (General Architecture for Text Engineering), which is an environment supporting the research and development of language processing software [9].

GATE provides a pipeline consisting of different component, each of which will handle a different part of the language processing. The components that are part of the pipeline, and come with GATE by default, are in order: Document Reset, ANNIE English Tokeniser, ANNIE Gazetteer, ANNIE Sentence Splitter and the ANNIE Part-Of-Speech Tagger.

Document Reset

The Document Reset component is used to reset the document, in this case a news item, to its original state. It will clear the document of all its current annotations. This enables the program to reannotate the text.

ANNIE English Tokeniser

With the tokeniser the text is split into very simple tokens, such as numbers, punctuation, and words of different types.

ANNIE Gazetteer

A gazetteer is a list with names of for example cities, countries, companies, days of the week, world leaders, etc. With the ANNIE Gazetteer words are looked up in the gazetteer lists in order to classify them. The work that is done by the ANNIE Gazetteer is limited in our implementation to some basic and static lists like days of the week, months of the year, etc. It is more preferable to use well defined rules in order to recognize special words. For example, a sentence like *"The conference will be attended by CEO's like*

Steve Ballmer and Steve Jobs", gives us the opportunity to recognize *Steve Ballmer* and *Steve Jobs* as CEO's. And because of the fact that we use news items, it is also possible to adapt the underlying ontology based on certain events. For instance, *"Steve Ballmer leaves Microsoft"*, informs us that *Steve Ballmer* is no longer the CEO of *Microsoft*.

ANNIE Sentence Splitter

With the Sentence Splitter, the text is split into sentences, which is required for the part-of-speech tagger.

ANNIE Part-Of-Speech Tagger

The ANNIE Part-Of-Speech Tagger is a modified version of the Brill taggger [5], which produces a part-of-speech tag as an annotation on each word or symbol [11]. The part-of-speech tags, like for example the ones described in table 3.1, can be used in the rules to describe certain patterns.

4.3 Rule Engine

After the preprocessing is completed the rules will be addressed. At the heart of the implementation is the Rule Engine. This component will provide the actual information extraction.

The rules are being processed by using a finite state automaton, which is often used in the field of language processing. An example of a deterministic finite state automaton that matches the simple grammar `(a|b)*abb` is shown in figure 4.1

In a similar manner, the text of each news article will be processed. Figure 4.2 shows an example of the finite state machine that matches rule 15. The tokens of a text are processed in the order in which they occur. For instance the sentence *Google introduces Chrome* will be processed in the following order. At the start, the finite state machine will be in state S_0. First the token *Google* will be examined. Since it matches the first transition the machine will move to state S_1. The second token, *introduces*, again matches the transition to S_2. Finally *Chrome* matches the last transition condition, which causes the automaton to move to S_3, which is a final state.

In case one of the condition to move to the next state (S_0, S_1, S_2 or S_3) is not met, the finite state machine ends up in the *Error*-state, which means that the current token-sequence doest match the rule.

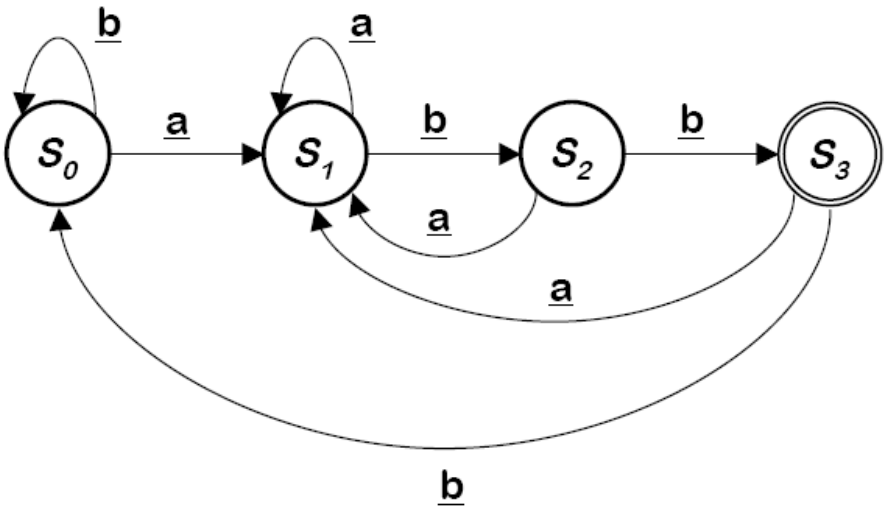

Figure 4.1: Deterministic Finite State automaton

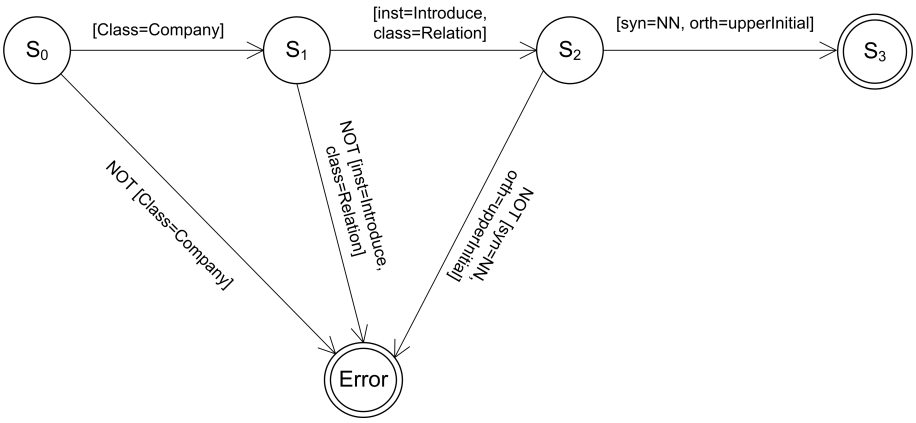

Figure 4.2: Finite State Automaton for Rule 15

4.4 Rule Editor

With the rule editor that we have created, knowledge engineers are able to construct rules that are used for the information extraction. Figure 4.3 shows the user-interface of the rule editor. The rule editor consists of three parts, the rule groups, the rules, and the editor.

4.4.1 Rule Groups

The rule groups can be used to denote a group of rules. In the extraction of information from text it is often the case that a set of patterns can be used to extract the same information. For example, the following two sentences

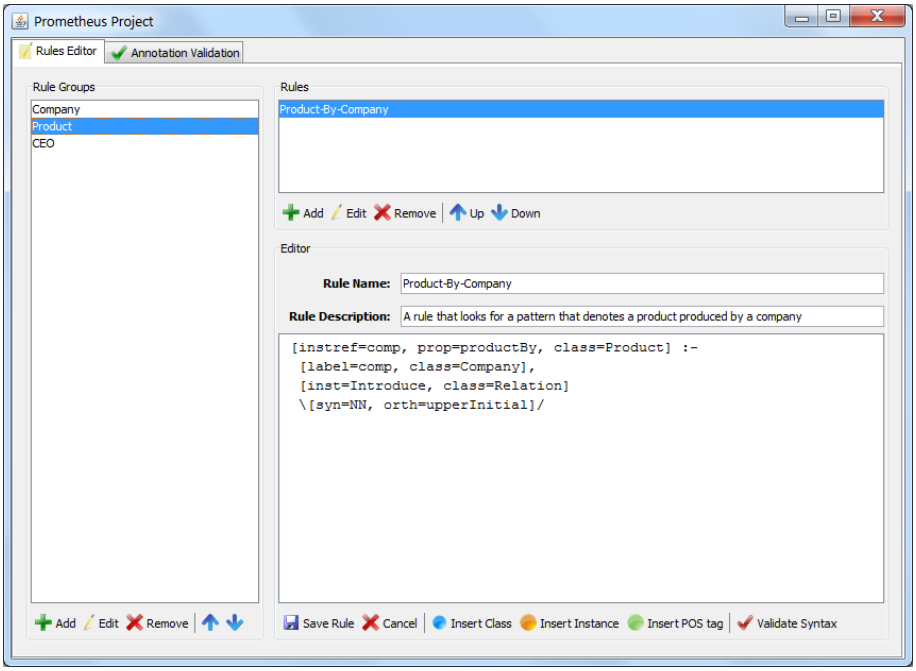

Figure 4.3: Rule Editor

both can be used to extract a product of *Apple*, however the pattern that should be looked for, is different:

Apple announces iPhone 3GS

Apple iPhone 3GS goes on sale today

4.4.2 Rules

Each Rule Group consist of one or more rules. Each rule describes a one or more patterns that are looked for in the text.

4.4.3 Editor

The interface finalIn addition we have provided the possibility to automatically check the syntax of the rule.

4.4.4 Rule Order

The order of the rules is important to extract the information in the right manner. For example, the rule shown in figure 4.3, requires that the company that occurs in the text should either exist in the ontology or in previous

annotations in order to be recognized as such in the text. This implies that the Company-rule should be run before the Product-rule. The order of the rules can be changed by using the blue arrow buttons at the bottom of the list with rule groups and rules.

4.5 Annotation Validation

After a news item leaves the GATE pipeline and the rules, that were created using the rule editor, are processed, the patterns that were identified will be listed in the Annotation Validator, which is shown in figure 4.4.

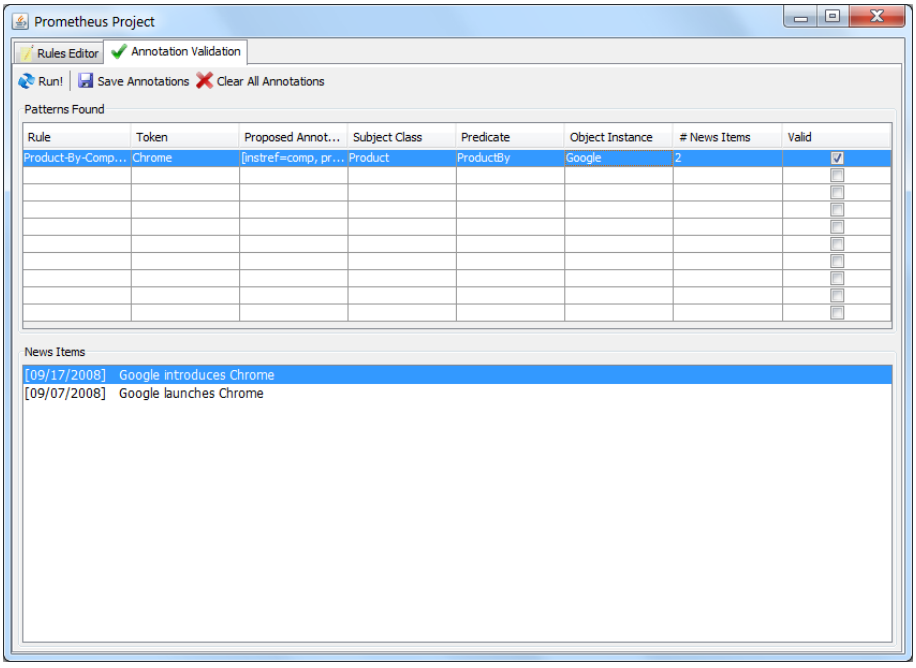

Figure 4.4: Annotation Validator

In this validation environment, the knowledge engineer or domain expert can review all the identified patterns. For each pattern that is found a set of properties is shown in the table. The first columns shows the rule that was matched, followed by the token (or set of tokens) that was found in the text and matched the rule. The third column contains the proposed annotation (e.g., the left-hand side of the rule) for the token that was found. The columns Subject Class, Predicate and Object Instance are optional and are only used when a the class of a token or an object property is to be annotated. The seventh column shows the number of news items in which the pattern was identified. Finally, the last column contains a check box that should be checked if the domain expert finds the identified pattern to be valid.

The list at the bottom of the user interface contains the news items (in this case only the titles) that belong to the selected pattern.

After the domain expert has verified the annotations that were found, they can be saved so they can be considered in the next run. For example, if in the first run the company rule was matched, the second run might identify a product that is developed by that company.

Clearing the news items of all annotations will remove all annotation that were found by the rule engine and then call the Document Reset from the GATE framework, which will make certain that all previous identified annotations, including tokenization, gazetteer lookups and part-of-speech tags, will be removed.

4.6 Conclusions

In this chapter we have shown the core components of the implementation of the rule based approach for information extraction from news items. The implementation is an extension to the Hermes News Portal, which is an Semantic Web based approach for personalizing news. We make use of the OWL domain ontologies that serve as the knowledge base containing news items and domain concepts with the corresponding relations.

We employ default GATE components to annotate tokens in text prior to the rule engine, that is based on a finite state machine to look for patterns described by the rules. The rules can be constructed by a knowledge engineer by using the Rule Editor, where the expert can define rule groups containing a set of rules that extract the same information. After the text is processed, the domain expert can use the Annotation Validator to validate the patterns that were found by the rule engine.

Chapter 5

Evaluation

In this chapter we evaluate the proposed approach of ontology learning by using extraction rules. To validate the results we examine two use cases and compare the extracted information by the rules with the information extracted by a domain expert.

5.1 Introduction

To evaluate the proposed approach we employ two use cases. For each use case we will construct a set of rules that looks for a certain pattern. We propose a very strict rule and a rule that is a little more relaxed and generic.

In the first use case we will create rules that try to identify new instances of certain classes. In the second use case we the rule helps in identifying a product that is created by a company.

5.2 Use Case 1: Discovering Classes

Many texts contain lists with tokens that belong to the same semantic class. If at least one of the tokens is already known in the ontology or annotated as such, it is possible to annotate the others in the list as well.

5.2.1 Headlines

The headlines for this use case are as follows:

Headline 1 Google revealed some official partners in the development of Chrome OS, such as Acer, Adobe, ASUS and HP.

Headline 2 Steve Jobs, Bill Gates and Freedom

Headline 3 What do Bill Gates, Steve Jobs, Steve Ballmer, Scott McNealy and Eric Schmidt have in common?

5.2.2 GATE Preprocess Results

First the headlines are being processed by the components in the GATE pipeline. After the news item exits the pipeline, each token is pre-annotated. For these headlines we are only interested in the lists with instances. Each token in such a list is either pre-annotated as a NNP (Proper Noun), a comma (,) or the word "and". With these annotations we create two rules.

5.2.3 Strict Pattern

With the first rule we assume that respectively *Acer* and *Bill Gates* are already present in the ontology, belonging to respectively the class *Company* and *BusinessLeader*.

Rule 18 tries to annotate new companies by looking for lists of companies. This requires the pattern to start with a token of the class *Company* followed by either a comma or the word "and". The part that will be annotated should have the syntactic category `NNP` and the orthographic category `upperInitial`. And this pattern should occur at least ones, to be able to capture both the first and last name.

```
[class=Company] :-
[class=Company], [token=","|"and"]
\ [syn=NNP, orth=upperInitial]+ /
```
(Rule 18)

Results

Each time the rule is run on the list with companies, it annotates one additional company. So, to annotate the whole list of companies it requires the text to be processed by the rule as many times as there are companies in the list. Each time it will extract one new company.

Headline 1 If we analyze the results produced by the GATE pipeline we see that the first headline correctly matches the pattern in the first run, because *Acer* is already known. So the result of the first run will be the annotation of *Adobe* as the class *Company*. The second and third run will also acquire *ASUS* and *HP* from the text.

Headline 2 The second headline will fail to match the pattern, because it does not contain a token which belongs to the class *Company*.

Headline 3 The same goes for the third headline, because this is also a list of business leaders instead of companies.

5.2.4 Relaxed Pattern

Rule 18 is strong but the strength fails to capture information that can be caught if we relax the rule. Rule 19 is changed in two ways. It makes use of the `classOf` feature, which refers to the class on the right-hand side of the rule that is labeled with the same value. Additionally on the right-hand side we us the OR-operator (denoted by "|") to also capture names that contain mixed capitals, such as *Scott McNealy*.

```
[classOf=prevItem] :-
[class=Company|BusinessLeader, label=prevItem],
[token=","|"and"]                                    (Rule 19)
\ [syn=NNP, orth=upperInitial|mixedCaps]+ /
```

Results

Headline 1 This rule still extracts the same information from the first headline.

Headline 2 The second headline will also succeed on the first run, because *Bill Gates* was already known in the ontology. On the second run, *Steve Jobs* precedes "Freedom". But *Freedom* does not belong to the class *BusinessLeader* even though it matches the pattern. So we can conclude that however the text matches the rule, it does not extract the correct information.

Headline 3 The third headline is correctly matched by this rule and annotates in four runs, all potential tokens with the class *BusinessLeader*.

5.3 Use Case 2: Company introduces Product

Financial news items often report about a company that introduces a new product. We can use information extraction rules to identify this product and annotate the relationship between the product and the company. In this use case we will list three headlines that contain information about a company, in this case *Google*, introducing a new product, *Chrome*. Subsequently, we analyze the headlines and show the annotations that resulted from the GATE pipeline. In section 5.3.3 and 5.3.4 we present a strict and a more relaxed rule and evaluate the effect of each rule on each headline.

5.3.1 Headlines

The headlines for this use case are as follows:

Headline 1: Google introduces shiny new browser prototype, Chrome.

Headline 2: Introducing Google Chrome.

Headline 3: Today, Microsoft presents its new browser, Gazelle.

5.3.2 GATE Preprocess Results

Again the headlines are first being processed by the components in the GATE pipeline. After the news item exits the pipeline, each token is pre-annotated. The partial annotations of the three headlines are shown in tables 5.1, 5.2 and 5.3.

Token	**Syntactic Category**	**Orthography**
Google	NNP	upperInitial
introduces	VBZ	lowercase
shiny	JJ	lowercase
new	JJ	lowercase
browser	NN	lowercase
prototype	NN	lowercase
,	,	
Chrome	NNP	upperInitial
.	.	

Table 5.1: Use Case 2 Headline 1 Pre-annotations

Token	**Syntactic Category**	**Orthography**
Introducing	VBG	upperInitial
Google	NNP	upperInitial
Chrome	NNP	upperInitial
.	.	

Table 5.2: Use Case 2 Headline 2 Pre-annotations

5.3.3 Strict Pattern

The first pattern assumes that *Google* and *Microsoft* already exist in the ontology and that they are known to belong to the class *Company*. In

Token	Syntactic Category	Orthography
Today	NN	upperInitial
,	,	
Microsoft	NNP	upperInitial
presents	VBZ	lowercase
its	PRP	lowercase
new	JJ	lowercase
browser	NN	browser
,	,	
Gazelle	NNP	upperInitial
.	.	

Table 5.3: Use Case 2 Headline 3 Pre-annotations

addition for the strict pattern we also presume that *Chrome* already exists in the knowledge base with as type *Product*. However the relation between *Google* and *Chrome* does not yet exist. The second product, *Gazelle*, is not yet present in the ontology.

On the right-hand side of Rule 20, first a token of the class *Company* is expected. Followed by an unspecified number of tokens, except for the dot (.), which would indicate the end of the sentence. Subsequently there should be an instance of *Introduce* of the type *Relation*. After which, zero or more unspecified tokens might follow. Finally the rule demands a token of type *Product*. This token will then be annotated with the annotation, specified by the left-hand side. Here we refer to the instance of *Company* found on the right-hand side and add the object property, *HasProduct*, to it with as subject the product found on the right-hand side.

```
[instref=company, objprop=HasProduct,
 class=Product] :- [class=Company,
 label=company], [token=*, syn!=.],
 [class=Relation, inst=Introduce], [token=*, syn!=.]
 \[class=Product]/
```
(Rule 20)

Results

Headline 1 With these assumptions in mind we run rule 20 on the first headline. The first token matches the first part in the rule. Then it is immediately followed by the verb "introduces" which is an instance of the class *Relation* and it also corresponds to the expected instance. Then the headline follows with some other tokens and ends with *Chrome*, which is known to be an instance of the class *Product*.

While the right-hand side of the rule matches this headline, the part between the backward and forward slash will be annotated as denoted by the left-hand side of the rule.

Headline 2 The first token in headline 2 will be ignored by the rule while the second token matches the rule. Then the rule expects any token except for the dot, which will succeed on the third token in the headline, but will fail on the last token. So the second headline does not match Rule 20.

Headline 3 This headline starts with *Microsoft*, which is known to be an instance of the class *Company*. Followed by "presents", which is corresponds to the instance *Introduce*. The rule fails however on the dot, while *Gazelle* is ignored, because it does not yet exist in the ontology as an instance of the class *Product*.

5.3.4 Relaxed Pattern

The results of the strict rule show that two of the three headlines did not match the rule. The second headline did not match, because the order of the words was different. This requires a completely different rule. The third headline however, failed because the last token was not yet known to belong to the class *Product*. This can be solved by taking a closer look at the annotations provided by the GATE pipeline. Table 5.1, 5.2 5.3 show that the product in all three headlines is annotated with the syntactic category `NNP` (Proper Noun) and the orthographic category `upperInitial`. So, instead of looking for a token with the class property set to *Product*, we adjust the rule to look for a token that has this syntactic and orthographic category, which results in Rule 21.

```
[instref=company, objprop=HasProduct,
 Class=Product] :- [class=Company,
 label=company], [token=*, syn!=.],
 [class=Relation inst=Introduce], [token=*, syn!=.]
 \[syn=NNP, orth=upperInitial]/
```

(Rule 21)

5.4 Conclusion

In the evaluation of the use of rules for information extraction we showed some advantages and a few shortcomings. First, the use of a strict rule makes it more likely to extract correct information, because the rule is specified to fit a very specific pattern. However, in case the text is a little different from the pattern, it will not be able to annotate it. Therefore we have used a

more relaxed rule to be able to match more sentences. These rules make use of OR-operators and different features, like `classOf`. We have also shown that the sometimes it is required to run rules more than ones to extract all the information. In addition to this, since each rule might extract new information that can be used by another rule, it is thus desirable to keep processing the text until there is no information left to extract by the rules as they are defined.

We may conclude that rules are very useful in the process of ontology learning. It requires the knowledge engineer to construct rules that are relaxed enough to extract as much information as possible, but strict enough to make sure that there are not too many false positives.

Chapter 6

Conclusions and Future Research

In this chapter we discuss our findings. In section 6.1, we answer the research question based on our literature review and the method we have developed and evaluated. In section 6.2 we discuss the future research.

6.1 Conclusions

We started with defining the following research question:

> *To what extent can we automatically discover and extract financial information in emerging news by using ontologies?*

In order to answer this question we will answer the following sub-questions:

1. How can we use patterns for information extraction?
2. What grammar can be used to describe information extraction rules?
3. How can we extend this grammar to employ ontologies in the information extraction process?

In the literature review we have shown that for a long time research is being done to the understanding of the natural language by computers. For human beings it is easy to understand and to disambiguate sentences when we have some basic knowledge of the language. Computers however, do not grasp the semantics of the sentences. It is therefore rather difficult to extract the right information from text. We have discussed some approaches that help automating this extraction, like concept induction, association rules and information extraction rules. In our approach we have focused on information extraction rules. Such rules define a certain pattern to look for in the text. Hearst [18] was one of the pioneers in the field and CAFETIERE [4]

followed the same concept. With patterns, only one instance of a specific relation is required, in contrast to statistical approaches that require a significant amount of instances, before a relation can be identified. Inspired by both, Hearst and CAFETIERE, we have defined our own syntax to describe information extraction rules.

In addition to the syntactic features, we also added semantics by employing ontologies in the description of the rules. We proposed to add features for classes, instances, object properties and datatype properties. By ordering the rules in a appropriate manner it is possible to annotate new instances of classes and add properties to them.

To facilitate the domain expert in constructing these information extraction rules, we have built a rule editor. In this editor the knowledge engineer can easily create and modify rule groups that contain a set of rules that help in extracting the same information. By first mining the news items with the default GATE components, like a tokeniser and a part-of-speech tagger, the text is already partially annotated before the rule-engine will process the rules. After the rules have been processed, the domain expert has to verify each pattern that was found in the text by using the Annotation Validator.

In the evaluation, we have highlighted the strengths and weaknesses of the rule-based approach for information extraction. Employing ontologies in rules, helps them to be created in a more generic manner. This way, different kinds of information may be extracted with the same rule. For example, one rule can be used in order to extract new companies as well as business leaders. We showed that with strict rules it is possible to extract a fair amount of information. This will probably require more rules to be able to extract a larger amount of information, since a strict rule focuses on one specific pattern in the text. While relaxing the rule will enlarge the amount of extracted information, the number of false positives will probably increase as well. To limit the number of false positives, the domain expert is required to verify each proposed annotation. We have also shown that each time the rule-engine processes the text, new information might be extracted, because one rule may depend on the information that was extracted by another rule in the previous run.

Despite of the fact that maintaining the rules and the validation of the annotations still requires a knowledge engineer, information extraction rules certainly help domain experts in extracting information and extending the knowledge base faster. With the help of ontologies, rules can be made more generic in order to extract different sorts of information.

6.2 Future Research

In this bachelor thesis we have done an evaluation based on use cases to point out the strengths and weaknesses of our approach. We propose to

extend this evaluation by creating a few rules and process a large amount of articles and compare the extracted information with the information that was extracted by a domain expert.

Besides doing a more extensive evaluation, our approach can be extended by updating the ontology. After the tokens in the text have been annotated, the annotations should be processed in order to update the ontology. A way to do this is by using update rules. Such a rule can be related to the extraction rule in order for the program to know how to proceed with the annotations provided by the rule engine.

At this point, rules have to be defined by a domain expert. This remains a rather tedious job. To overcome this, it would be interesting to automatically learn new patterns. For instance, if *Google* is known to be an instance of the class *Company* and *Chrome* of the class *Product* and they are known to be related to each other, by mining text, new patterns can be discovered. These patterns can then be used to discover new instances of the class *Company* and *Product* with the same type of relationship.

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation 10 February 2004, 2004. from: http://www.w3.org/TR/owl-ref/.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[4] William J. Black, John McNaught, Argyris Vasilakopoulos, Kalliopi Zervanou, Babis Theodoulidis, and Fabio Rinaldi. CAFETIERE: Conceptual Annotations for Facts, Events, Terms, Individual Entities, and RElations. Technical Report TR-U4.3.1, Department of Computation, UMIST, Manchester, January 2005. from: http://www.nactem.ac.uk/files/phatfile/cafetiere-report.pdf.

[5] Eric Brill. A Simple Rule-Based Part of Speech Tagger. In *Applied Natural Language Processing*, pages 152–155, 1992.

[6] P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology Learning from Text: An Overview*. IOS Press, 2005.

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

[8] Philipp Cimiano and Steffen Staab. Learning by Googling. *SIGKDD Explorations*, 6(2):24–33, December 2004.

[9] H. Cunningham. GATE, a General Architecture for Text Engineering. *Journal Computers and the Humanities*, 36(2):223–254, May 2002. from: http://gate.ac.uk/.

[10] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Technical Report CS–00–10, University of Sheffield, Department of Computer Science, 2000.

[11] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. *Developing Language Processing Components with GATE Version 5*. http://gate.ac.uk, 2009.

[12] Oren Etzioni, Stanley Kok, Stephen Soderland, Michael Cafarella, Ana-Maria Popescu, Daniel S. Weld, Doug Downey, Tal Shaked, and Alexander Yates. WebScale Information Extraction in KnowItAll (Preliminary Results). In *Proceedings of the 13th International World Wide Web Conference*, 2004.

[13] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[14] Flavius Frasincar, Jethro Borsje, and Leonard Levering. A Semantic Web-Based Approach for Building Personalized News Services. *International Journal of E-Business Research (IJEBR)*, 5(3):35–53, 2009.

[15] Tom Gruber. What is an ontology?, 2002. from: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.

[16] Tom Gruber. Ontology. *Encyclopedia of Database Systems*, 2008. (to appear).

[17] Marti A. Hearst. Automatic aqcuisition of hyponyms from large text corpora. In *Fortheenth Internation Conference on Computational Linguistics*, July 1992.

[18] Marti A. Hearst. Automated Discovery of WordNet Relations. *WordNet: an electronic lexical database and Some of its Applications*, pages 131–151, 1998.

[19] Paul S. Jacobs, George R. Krupka, and Lisa F. Rau. Lexico-Semantic Pattern Matching As A Companion To Parsing in Text Understanding. In *Fourth DARPA Speech and Natural Language Workshop*. Morgan-Kaufmann, 1991.

[20] Jorg-Uw Kietz and Katharina Morik. A Polynomial Approach to the Constructive Induction of Structural Knowledge. *Machine Learning*, 14:193–217, 1994.

[21] Alexander Maedche and Steffen Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, March 2001.

[22] Brian McBride. Jena: Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002. from: http://jena.sourceforge.net.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, March 2005. from: http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/.

[24] Alan Turing. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, October 1950.