

RULE BASED SEMI-AUTOMATIC ONTOLOGY LEARNING

J.A. Borsje B.Sc.
`research@jborsje.nl`

Master Thesis

Economics and ICT, Informatics and Economics
Econometrics Institute
Erasmus School of Economics
Erasmus University Rotterdam

Supervisor: Dr. Ir. F. Frasincar
Co-supervisor: Dr. W. Pijls
Internship supervisor: Dr. Ir. M.P.W. Vreijling

18th July 2007

Abstract

Ontologies are the backbone of the Semantic Web. They are used to formally describe the meaning of terms in a specific domain. However, the world around us is changing quickly. These changes should be reflected in the domain specific ontologies. Manually updating these ontologies is very time-intensive, and therefore very expensive, because it requires the constant attention of domain experts and knowledge engineers. A (semi-)automatic ontology learning framework can be used to significantly reduce the amount of time needed to update ontologies.

We propose a rule based ontology learning framework to update the ontology. A rule consist of a lexico-syntactic pattern, which has syntactic arguments based on ontological classes, and one or more actions which should be executed once the pattern is found. Using these lexico-syntactic patterns we mine news items in order to find occurrences of events. Based on the identified events the actions are executed, thereby updating the ontology.

Keywords: Semantic Web, Ontology Learning, Natural Language Processing, Data Mining, and Knowledge Representation.

Acknowledgements

I would like to thank a few people who have helped me out during the process of writing this thesis. First and foremost I would like to thank my primary supervisor Flavius Frasinca for his insights and support. He was a very important source of inspiration.

Secondly, I would like to express my gratitude to my colleagues at SemLab, who have helped me during my internship. They made the writing of this thesis a fun and interesting experience. I especially want to mention Mark Vreijling, who guided me during my internship, and provided me with a lot of good ideas. I also want to thank Job Tiel Groenstege and Trude Gentenaar for their practical help in the field of NLP. Besides that I want to thank Richard van Duijn for his help with software architectural issues.

Last, but certainly not least, I would like to thank my girlfriend and my parents. They have supported me all the way, during the writing of this thesis. Besides that, my parents have always encouraged me to keep on learning and I am very grateful for that.

– Jethro Borsje, 30 June 2007

Contents

List of Figures	v
List of Tables	vi
1 Introduction	1
1.1 Background	2
1.2 The Problem	3
1.3 Goal	4
1.4 Methodology	5
1.5 Structure	6
2 Related Work	8
2.1 Natural Language Processing	8
2.1.1 Introduction	8
2.1.2 Stanford Grammar Parser	9
2.1.3 General Architecture for Text Engineering (GATE)	9
2.1.4 OpenNLP	10
2.1.5 MontyLingua	10
2.1.6 Conclusion	10
2.2 Ontology learning	11
2.2.1 Introduction	11
2.2.2 Pattern Matching	12
2.2.3 Co-occurrence Analysis	13
2.2.4 Hybrid Approach	14

2.2.5	Hierarchies in Text	14
2.2.6	Ontology Learning Projects	15
2.2.7	Human Intervention	16
2.2.8	Conclusion	16
3	Rule Syntax	18
3.1	Introduction	18
3.2	Patterns	19
3.3	Actions	22
3.3.1	Use Cases	22
3.3.2	Generalizations	23
3.3.3	Alternatives	24
3.3.4	Conclusion	27
4	Rule Engine	29
4.1	Introduction	29
4.2	Rule Editor	30
4.3	Event Detector	32
4.3.1	Introduction	32
4.3.2	GATE Pipeline	35
4.4	Event Validation	38
4.5	Action Execution Engine	39
5	Evaluation	41
5.1	Introduction	41
5.2	Experimental Setup	42
5.2.1	Introduction	42
5.2.2	First Run - Strict Pattern	42
5.2.3	Second Run - Relaxed Pattern	43
5.2.4	Third Run - Hybrid Pattern	43
5.2.5	Reference Standard	43
5.3	Measures	44

5.3.1	Introduction	44
5.3.2	Accuracy	45
5.3.3	Error	46
5.3.4	Precision	47
5.3.5	Recall	47
5.3.6	F_1	47
5.3.7	Usefulness	47
5.4	Results	48
5.4.1	First Run - Strict Pattern	48
5.4.2	Second Run - Relaxed Pattern	49
5.4.3	Third Run - Hybrid Pattern	50
5.5	Conclusion	51
6	Conclusions and Future Research	54
6.1	Conclusions	54
6.2	Future Research	55
6.2.1	Entire Documents Processing	56
6.2.2	Event Chains	56
6.2.3	Automatic Pattern Discovery	56
	Bibliography	57

List of Figures

2.1	The Ontology Learning Layer Cake.	12
3.1	Competitor SWRL rule.	24
3.2	Competitor SWRL rule using the differentFrom property.	25
3.3	Competitor SWRL rule using negation.	25
3.4	Simple SPARQL remove query.	26
3.5	Simple SPARQL add query.	26
3.6	Advanced SPARQL remove query.	27
4.1	The rule editor, showing the interface for editing relations.	31
4.2	The rule editor, showing the interface for editing rules.	31
4.3	The action editor.	32
4.4	The validation environment of the rule engine.	39
4.5	Abstract action before substitution.	40
4.6	Action after substitution.	40

List of Tables

5.1	The symbols used in the measures.	44
5.2	The contingency table.	45
5.3	The data when running the rule-engine using the strict pattern. .	48
5.4	The data when running the rule-engine using the relaxed pattern.	49
5.5	The data when running the rule-engine using the hybrid pattern.	50
5.6	An overview of the measures for the three runs.	52

Chapter 1

Introduction

Personal beauty is a greater recommendation than any letter of introduction.

– Aristotle, 384 BC - 322 BC

This chapter provides an introduction to this thesis. Section 1.1 discusses the Semantic Web project and the use of ontologies.

Section 1.2 describes the problem that the real world is changing constantly, and the fact that these changes should be reflected in ontologies.

In section 1.3 the goal of this thesis is outlined. We identify the need for an ontology learning framework which takes expert knowledge into account. Based on this assumption a research question is formulated.

Section 1.4 discusses the methodology followed during the writing of this thesis. Based on the research question three hypotheses are defined. We focus on the different research activities, and on the various tools and languages that we use.

Finally, we present the structure of the thesis in section 1.5. This gives a clear overview of the way the thesis is organized.

1.1 Background

At this moment the Internet provides users with a large amount of mostly unstructured information. The Semantic Web [1] project of the World Wide Web Consortium (W3C) provides a framework that allows for data to be shared and reused. It is an effort to create common formats to structure data so that it will be interchangeable and machine understandable. One of these formats is the Web Ontology Language (OWL) [2], which is an ontology language that can formally describe the meaning of terms. When data is properly annotated using OWL, a computer can reason with it, making it easier for computers to integrate the data with other information.

Semantic Web technologies can be used in numerous different ways. An example application is the generation of an ontology which contains domain specific knowledge. This ontology can contain all kinds of concepts and relations between these concepts relevant for a specific domain, say the pharmaceutical stock market. Using such an ontology, domain knowledge can be interchanged between members of a community or employees of an enterprise. If an ontology is well-structured, it can be used for relation-based-searching. This means that questions of the following nature can easily be answered: “Who are the competitors of Akzo Nobel?”, “Which drugs are used when one has diabetes?”, “What are the products of Bausch & Lomb?”, etc.

These domain specific knowledge bases in the form of ontologies can be used for the classification of news items. Ontology concepts to which the news items are related can be found by mining the news items for the occurrence of lexical representations of these concepts. When the relations between the concepts in the ontology and the news items are also stored in an ontology, users can easily retrieve news items which are related to relevant ontology concepts. Questions of the following nature can then be resolved: “Give me all the news related to competitors of Akzo Nobel”, “Give me the news of the past two weeks, related to all the products of Johnson & Johnson”, etc.

Such an approach enables users, in this case pharmaceutical stock market

analysts, to systematically review news items. They no longer have to search through an enormous amount of data before they find what is relevant to them. This saves a lot of time and enhances the quality of the analysis of the stock market by providing the analysts with relevant information. However, the quality of the news item classification process largely depends on the quality of the knowledge stored in the domain specific ontology.

1.2 The Problem

The world around us is changing quickly, new products, markets, and businesses are founded every day. All these changes need to be reflected in the knowledge base, in order to keep it as up to date as possible thereby enabling the creation of useful Semantic Web information systems [3]. Because the world around us is changing so rapidly it is a time consuming job to maintain these ontologies. They would be more useful if they could somehow be maintained (semi-)automatically. Various attempts to maintain ontologies (semi-)automatically have been made [4, 5], making use of external knowledge bases such as WordNet [6], SemCor [7], online dictionaries, etc.

Most of these approaches use text mining techniques to extract terms which are then disambiguated using lexical databases such as WordNet [8]. Although these approaches can achieve fairly good results, they do not make use of the expert knowledge of the human domain experts. They mostly identify relations between concepts based on some lexico-syntactic rules, which means that expert knowledge is not explicitly used. As has been argued by [5, 8, 9], (at least a part of) the results found by these approaches must still be validated by a domain expert.

Many scientists have found that patterns are very useful to find relations between concepts in unstructured text [9, 10, 11]. The found relations include all kinds of lexico-syntactic relations such as: hypernym, hyponym, meronym, holonym, etc. Using these relations, concept hierarchies can be derived, e.g. superclasses, subclasses, siblings, etc. [12]. However the automatic learning of

hierarchies is not enough to create a valid, relevant, and accurate ontology.

1.3 Goal

To make use of the benefits of the emerging Semantic Web technologies, domain knowledge must be gathered into a structured ontology. Creating, and especially maintaining, these domain specific ontologies is a time consuming job which must be done by knowledge engineers and domain experts. The goal of this master thesis is to provide a framework which facilitates semi-automatic ontology learning through semantic rules created by domain experts, thus making use of expert knowledge. These semantic rules define lexico-syntactic patterns which can be used to mine news items for the occurrence of specific events, and actions which must be executed once an event is found.

The research question of this thesis is:

How to exploit domain knowledge in a pattern-action based approach for ontology learning?

The proposed usage of patterns has similarities with other approaches like [9], but the introduction of expert-defined-actions is a novelty. This does not only enable an ontology learning mechanism to find a certain relation based on a pattern, but it also allows for the execution of one or more specific actions once the pattern is found. Take for example the following sentence:

“Roche agrees to buy Therapeutic Human Polyclonals”

An example pattern would be:

[Company] buys [Company]

Using this pattern the following relation can be extracted:

Roche $\xrightarrow{\text{buys}}$ Therapeutic Human Polyclonals

This relation can be added to the ontology, but more advanced consequences of this news can not be inferred without using expert knowledge. For example: the products of Roche and Therapeutic Human Polyclonals do no longer compete with one another. An expert might define the action that, once the previously discussed “buy pattern” is found, the products of the two respective

companies will no longer compete with one another. Enabling domain experts to define patterns with accompanying actions facilitates the generation of an ontology learning tool, which can perform advanced actions once simple patterns are found, thereby facilitating cause-and-effect reasoning.

1.4 Methodology

To answer the research question as presented in section 1.3 of this document, we define three different hypotheses:

1. Expert knowledge is needed for facilitating ontology learning.
2. A formal rule syntax is useful when formalizing expert knowledge.
3. By using an expert defined rule base, an ontology learning mechanism can be created.

To draw conclusions on these hypotheses, a few different methodologies have to be used. This will be done consecutively and they will result in different chapters in this thesis. The first methodology that will be used is a literature review. Using existing literature, current approaches will be evaluated to see if they are useful for answering the research question. Using this literature review the first hypothesis is checked for validity.

After the literature review a formal rule syntax is constructed which facilitates the formulation of rules by domain experts. A small study into SWRL [13] is done, to see if it is applicable in this case. These rules are used by an event detector which mines news items, looking for the occurrence of events based on the defined lexico-syntactic patterns. If an event is found, the actions associated with it are executed, thereby semi-automatically extending the ontology. This approach is used to check the validity of the second and third hypotheses.

To construct the event detector, some existing Java libraries will be used. At this moment two different sets of libraries are needed, one for using OWL ontologies in Java, and another for Natural Language Processing (NLP). The Jena API [14] is used to facilitate the processing of OWL ontologies in Java. For

the NLP task there are several libraries which can be used such as GATE [15], OpenNLP [16], MontyLingua [17], etc. These will be discussed in more detail in section 2.1.

To evaluate the results of the learning process, an expert should manually look at some news items and identify which events, if any, can be found in these news items. These manually identified events can then be compared with the results of the semi-automatic learning process [18]. This will be done to be able to draw conclusions about the precision and recall of the rule engine. Other metrics, such as discussed in [19] can also be used in this validation process.

1.5 Structure

The remainder of this thesis is organized as follows.

In chapter 2 the work related to the problem being addressed is discussed. A few different approaches to natural language processing (section 2.1) and ontology learning (section 2.2) are reviewed. For each approach the strengths and weaknesses are evaluated. We will also assess to what extent the discussed approaches are useful with respect to our goal.

Chapter 3 is divided in three sections and focuses on the formulation of a rule syntax. Section 3.1 introduces the concept of rules and explains why they are needed. After that, section 3.2 focuses on the patterns which are embedded in the rules. It discusses the requirements of these patterns together with their advantages. Finally, section 3.3 describes the actions which can be used with the rules.

After that, chapter 4 focuses on the construction of a rule engine, which is introduced in section 4.1. The rule engine contains an editor to create rules (discussed in section 4.2), and an event detector which is capable of parsing news items with respect to the formulated rules. The way events are found in text based on lexico-syntactic patterns using the event detector is described in section 4.3. After events are located the appropriate actions are to be executed. The procedure of executing the actions is discussed in section 4.5.

The subject of chapter 5 is the evaluation of the results of the learning process. To validate the results we compare the events identified by the rule-engine with those identified by a domain expert, based on the same set of news items.

In chapter 6 we present our findings. Based on our results the validity of our earlier defined hypotheses are tested. We also give an answer to the formulated research question. In the end we formulate our conclusions and identify future research directions.

Chapter 2

Related Work

Mi taku oyasin. (We are all related.)

– Lakota belief

A lot of research has already been done in the fields of Natural Language Processing (NLP), and ontology learning. This chapter provides an overview of this research, and outlines the strengths and weaknesses of several approaches.

2.1 Natural Language Processing

2.1.1 Introduction

One of the first NLP applications was called ELIZA, it was programmed by Joseph Weizenbaum in the 1960s [20]. Its functioning was based on identifying keywords, which were linked to certain reply phrases. Based on the input of the user, ELIZA could generate an answer that fitted into the conversation. ELIZA applied a template to every sentence in order to construct a fitting answer. Using this template, sentences were broken up in smaller parts which were then analyzed and classified. These classified parts would then be used in examining the structure of sentences. Based on this structure a fitting answer could be created. The field of NLP has come a long way since then, although ELIZA remains a good example of how NLP basically works. Nowadays there are a lot

of different software packages available in the area of NLP. These packages range from simple grammar parsers to complete NLP development environments.

For our research we primarily need a parser which is capable of parsing (English) text in order to determine the grammatical structure of sentences. This structure is needed to perform useful analysis on text, based on which ontology learning can be achieved. Besides grammar parsing we need basic I/O (input/output) facilities for processing text, and we have to be capable of handling ontologies. Besides that packages that are implemented in Java are preferred, because they are easy to integrate in already existing software packages like Jena [14], and ARQ [21]. We will now describe a few NLP packages in more detail in order to be able to make an informed decision on which package to use in this research project.

2.1.2 Stanford Grammar Parser

The Stanford grammar parser is a statistical parser which is available in several languages including English, German, and Chinese. The Stanford grammar parser is implemented in Java. It is capable of parsing and tagging sentences which results in a parse tree of POS (Part-Of-Speech) tagged words [22, 23]. Although the results of this package are quite nice, they are very basic. This tool does not provide us with any more functionality than just parsing and tagging. The package does not use external knowledge bases such as gazetteer lists or ontologies to identify domain specific terms.

2.1.3 General Architecture for Text Engineering (GATE)

The *General Architecture for Text Engineering* (GATE) [24, 15] is a well known text annotation framework, containing both a Software Development Kit (SDK) and a graphical development environment. It is a modular system which provides a lot of out-of-the-box functionality which otherwise would have to be made from scratch. A nice example of this are the I/O facilities that are embedded into the GATE framework. It is able to read (and write) several different document formats such as RTF, HTML, and XML. It also support the use of

OWL ontologies through the use of specialized modules.

Due to its modularity GATE is very extensible, because developers can create their own modules which can then be used together with the existing GATE modules to create an application. GATE also comes with its own rule language called JAPE (*Java Annotation Patterns Engine*) [25] which can be used to formulate rules using annotation based patterns which use regular expressions. Other advantages of GATE are the fact that it is implemented in Java, it is actively being developed, and it has a very active user community.

2.1.4 OpenNLP

The OpenNLP project [16] aims to provide a fairly complete framework of separate Java packages which can perform several different commonly used NLP tasks. It includes Java packages for sentence detection, tokenization, POS-tagging, parsing, etc [26]. This framework looks promising, however the development seems to have stopped.

2.1.5 MontyLingua

MontyLingua is a software package which is capable of extracting subject/verb/object tuples, adjectives, noun phrases, verb phrases, etc. [17, 27] It basically performs POS-tagging on top of which some extra analysis is done in order to provide users with a semantic interpretation of the text. The package is available both in Python and Java, however it has not been updated for over two and a half years.

2.1.6 Conclusion

After careful evaluation and experimentation we have decided that GATE is the most appropriate NLP software package to be used in this research. The biggest advantage of GATE is its modularity. It comes with a lot of different modules which can be used in various combinations in order to achieve user-tailored NLP. Besides the out-of-the-box modules GATE is easily extensible with user-written modules, because of the clear Java interface. In addition to that it provides

users with the ability to formulate their own JAPE rules which can be used in the NLP process.

Although GATE is a complex software package which is hard to work with at the start, it is very useful because of the feedback provided by its active user community, and extensive documentation. GATE is still actively being developed which is very important, because this research uses cutting edge technologies in the field of the Semantic Web. At this point the GATE development team is busy with adjusting the GATE ontology API in order to facilitate advanced ontology usage. Another important advantage is the fact that GATE is capable of handling several document formats as both its input and output. This is opposed to the other previously discussed software packages which are mostly only capable of handling raw textual data. A final advantage lies in the fact that GATE is widely accepted in both the scientific community, and the business world. There are several commercial applications using GATE such as the h-TechSight Knowledge Management Portal [28] made by OntoText.

2.2 Ontology learning

2.2.1 Introduction

Ontology learning has become a popular field of research since the emergence of the Semantic Web. Before discussing several ontology learning approaches the terms *ontology* and *ontology learning* have to be defined.

An *ontology* is a specification of a conceptualization [29]. It consists of two separate layers called a T-Box (Terminological-Box) and an A-Box (Assertion-Box) [30, 31]. T-Box statements describe a domain in definitional terms, such as a set of classes and properties. A T-Box can be seen as a terminological vocabulary. A-Box statements are facts associated with a T-Box, they can be seen as instances of classes and properties. It is common practice to refer to the T-Box as the actual ontology while the A-Box is called a knowledge base. However, in this thesis we refer to both the T-Box and the A-box when using the term *ontology*.

Ontology learning refers to the tasks associated with creating and maintaining a domain specific ontology. However, in the ontology learning community there is no consensus on what these tasks exactly are. In [32] an ontology learning layer cake is used to classify several different ontology learning tasks. A modified version of this layer cake is depicted in figure 2.1 (derived from [12]). This layer cake visualizes the different aspects of ontology development in an increasing order of complexity.

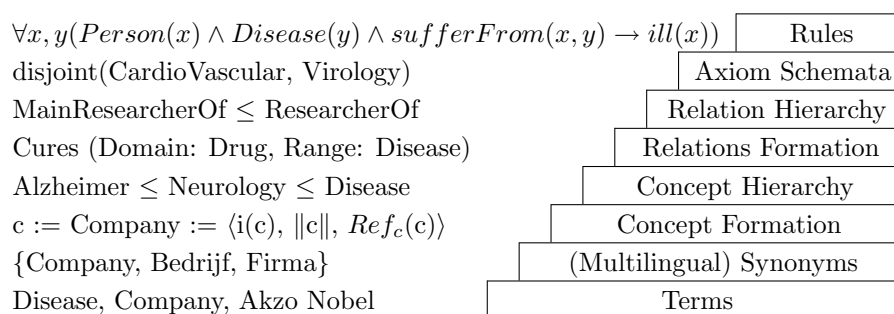


Figure 2.1: The Ontology Learning Layer Cake.

This layer cake shows how ontologies are created and populated, based on low-level terms. Our approach uses semantic rules to identify events in news items. Based on these events instances of concepts are created, and modified.

In this section we discuss several approaches, thereby giving a brief overview of the research in the field of ontology learning.

2.2.2 Pattern Matching

Lexico-syntactic pattern based information extraction has been proposed by Hearst in [9, 33]. This approach uses an existing ontology to extract pairs of related concepts in order to find hyponym and hypernym relations. This is done by finding regular expression patterns in free text.

This approach has been implemented in several research projects, such as [12] in which the original lexico-syntactic pattern matching methodology of Hearst is extended with a clustering algorithm. This algorithm places similar terms

in clusters in order to be able to label these similar terms with a proper label. These clusters are created based on a metric for item similarity. Using these clusters classes can be created which are populated with the terms as their instances.

Another use of the lexico-syntactic patterns of Hearst can be found in [34] which combines their Ontology Refinement [35] system [36] with the approach of Hearst. The system uses word distance [37] in order to determine where a new synset has to be placed in the WordNet ontology. The patterns of Hearst are used when the context of the new synset alone is not enough to determine where the new synset has to be placed. The research showed that the accuracy of both the Ontology Refinement system and the hyponymy pattern approach benefit from one another.

2.2.3 Co-occurrence Analysis

Co-occurrence analysis is used to analyze the co-occurrence of words in a corpus of text. This basically means that an analysis is conducted into which words co-occur frequently. As has been stressed by [38] there is a semantic similarity between words that co-occur together frequently in similar linguistic contexts. This implies that there is an associative relation between words that co-occur together, which is of particular interest because it explains the reason why words are co-occurring. If these associative relations can be learned (semi-) automatically, ontology learning can be partly achieved

One approach is to find semantically related words based on co-occurrence in an automated fashion, these can then be labelled manually with the appropriate relation [39]. In [40] co-occurrence is used to retrieve words that co-occur frequently after which NLP techniques are used to determine the relation between the co-occurring words.

Co-occurrence is used in many different ways, an example of this can be found in [41]. This approach uses co-occurrence to calculate the semantic distance between words. Based on this semantic distance they determine which words describe which concepts. Another example can be found in [42]. This approach

looks at the word correlation in different scopes, being sentences, paragraphs, and documents. These different scopes are then used to investigate the influence of different scopes of co-occurrence on the quality of the derived concepts.

Another approach uses the tree like structure of lexical taxonomies as a decision tree which can be used in the extension process of these taxonomies. This approach can be adapted to work for ontologies, because ontologies can also be depicted as a tree like data structure [43]. Co-occurrence is used to determine the similarity between a new concept, and already known concepts. Based on this similarity the place at which to insert the new concept can be determined.

Latent Semantic Analysis (LSA) [44, 45] is another technique which is based on occurrence. Using LSA latent semantic relations (i.e. non-explicit semantic relations) can be discovered. LSA assess similarities across documents in order to find different occurrences of the same concept, based on the context in which they appear. This approach is used to find similar concepts (i.e. concepts with the same semantic meaning), although they are not mentioned in the exact same way.

2.2.4 Hybrid Approach

Some approaches, like [46], use both pattern matching and co-occurrence or statistical analysis. This approach first uses pattern matching in order to extract relevant pieces of text out of a large corpus of documents. The result of this phase is statistically processed in the second phase. In this second phase the most relevant domain specific terms and relations are extracted, thereby filtering out more general terms.

2.2.5 Hierarchies in Text

Another approach to learning relations between concepts looks at the hierarchy of structured text. Scientific and technical documents are usually organized in a certain hierarchy. Moreover, documents themselves are also organized in a hierarchical way, papers are organized in conferences or journals, chapters are

organized in books, etc. This hierarchical structure can be exploited by an algorithm in order to learn subordinate relations [47].

2.2.6 Ontology Learning Projects

In recent years several attempts have been made to transform abstract ideas about ontology learning into concrete and useful applications. One of these applications is called PANKOW (Pattern-based Annotation through Knowledge on the Web) [19]. PANKOW uses the Google™ API to search the web for occurrences of specific lexico-syntactic patterns. By gathering collective knowledge about a term through searching the web, an ordered list of suggestions about the meaning of a term can be created. PANKOW then shows this ordered list of suggestions to a domain expert, who can use the collective knowledge together with his individual knowledge, and the context in which the terms appears. Based on the collective knowledge, his individual knowledge, and the context of the term a domain expert can accurately annotate a new term. PANKOW can also be used to learn sub-/superconcept relations [19].

Another concrete ontology learning application is OntoLearn [8, 48] which is used to automatically enrich WordNet. Using OntoLearn domain specific concepts can be added to WordNet, and WordNet glosses can be disambiguated. OntoLearn uses a Word Sense Disambiguation (WSD) algorithm called Structural Semantic Interconnection (SSI) in order to determine how the new concept is related to the concepts in the ontology.

SemNews [49] is an application which monitors RSS news feeds in order to provide a structured representation of the meaning of news items. SemNews performs natural language processing on the incoming news items after which it extracts new instances from the text. These new instances are found based on the result of the natural language processing, an ontology which contains over 8K of concepts, and an onomasticon [50] (a lexicon of proper names) which contains over 400K of terms.

2.2.7 Human Intervention

As has been stated by [51], human intervention is needed to supervise the ontology learning process. The reason for this is that natural language processing is still error prone, therefore humans are needed to validate the result. However, natural language processing can be used to prune large corpora of texts, thereby extracting only relevant information. Based on this relevant information suggestions can be made on how to extend the ontology. These suggestions should then be validated by domain experts, thereby creating a semi-automated ontology learning framework.

2.2.8 Conclusion

In this section we gave a brief overview of different ontology learning mechanisms, a more complete overview of research in this field can be found in [52].

In our work we will develop a pattern based learning framework, inspired by the approach as proposed by Hearst which we will extend with ontology specific patterns. This extension is needed because we feel that the patterns as proposed by Hearst are too general to yield good and practical results. Because we want to (semi-)automatically extend domain specific ontologies we feel that domain specific rules are required, thereby reusing existing knowledge.

Co-occurrence learning is an interesting approach which might be used in the future as an extension of our rule based ontology learning framework.

The hierarchical learning approach is not useful to us, because we aim to be able to learn from unstructured text such as news items in RSS feeds. A characteristic feature of news items in news feeds is that they are not very structured. They mainly consist of a few lines of text which very briefly tells the reader what the news is about. Chapters, sections, and paragraphs are not used in these news items.

As has been sketched by various researchers, complete unsupervised ontology learning is a non-trivial task. We believe that it is not yet possible to achieve full unsupervised learning, due to limitations in the fields of both NLP and artificial

intelligence. Therefore we propose a framework in which domain experts have to validate the findings of the algorithm. In this framework these domain experts can formulate patterns which are used to identify the occurrence of specific events. Using these patterns large corpora of texts can be parsed in order to extract the most relevant textual items. Based on the identified events advanced expert-defined actions can be used to achieve the actual learning.

Chapter 3

Rule Syntax

The golden rule is that there are no golden rules.

– George Bernard Shaw, 1856 - 1950, Man and Superman (1903)

“Maxims for Revolutionists”

This chapter focuses on the formulation of a rule syntax. Section 3.1 introduces the concept of rules and explains why they are needed. After that, section 3.2 focuses on the patterns which are embedded in the rules. It discusses the requirements of these patterns together with their advantages. Finally, section 3.3 describes the actions which can be used with the rules.

3.1 Introduction

Our ontology learning framework uses rules to find events, and executes actions based on these events. This allows domain experts to use cause-and-effect reasoning in the ontology learning process. An event (one company buys another company) causes one or more changes in reality (the products of the two companies do no longer compete with one another, the bought company ceases to exist in its old form, etc.). Using our rule syntax domain experts can model these events and the resulting changes, thereby creating an ontology learning system which is able to learn from the occurrence of specific events.

We make use of an OWL ontology to store the rules, because this enables easy integration with already existing ontologies. We have formulated a rule syntax, which facilitates the use of classes from other ontologies in the construction of lexico-syntactic patterns defining the events. A rule consists of two parts: a *pattern* (which can have multiple lexical representations), and one or more *actions* (which can be executed once the pattern has been matched). In terms of proposition logic the pattern is the antecedent, and the actions are the consequents.

In this chapter we describe the syntax of patterns, and actions. In chapter 4 the actual implementation and use of this syntax in a software package is discussed.

3.2 Patterns

The lexico-syntactic pattern of a rule is used to mine text for the occurrence of a specific event. Such a pattern consists of a *subject*, a *relation*, and an optional *object*. The *subject* and the *object* are the syntactic arguments of the *relation*, they are describing the possible participants in the event. In our implementation the *subject* and *object* are OWL classes, which reside in the ontology that needs to be extended by the learning process. The OWL individuals (i.e. instances) of these classes are the possible participants in the event. The *relation* is an OWL individual of the predefined OWL class “Relation”. The *object* is optional, because there are situations in which only a *subject* and a *relation* are enough to trigger an *action*. An example of such a pattern is:

[Company] goes bankrupt

From one pattern multiple lexical representations describing the same event can be derived, which are used in the pattern matching process. This is done in several consecutive steps. The first step retrieves the pattern, which is associated with a specific rule. The second step substitutes all the semantic classes by the participants which they describe (i.e. their instances), with the limitation that a participant can not be a subject and an object at the same time. The third

step substitutes both the participants (subject and object), and the relation for all the lexical representations by which they are denoted.

We will illustrate this process with the following example pattern, which we call the “buy-pattern”:

[**kb:Company**] kb:buys [**kb:Company**]

In this pattern [**kb:Company**] is the URI of a class in an OWL ontology, kb:buys is the URI of an individual of type “Relation” in this ontology. After this pattern has been retrieved from the rules ontology the second step of the process replaces [**kb:Company**] for all instances of class “Company”. If there are three companies in the ontology (being: kb:Roche, kb:JohnsonAndJohnson, and kb:AkzoNobel), the following patterns will be created:

- kb:Roche kb:buys kb:JohnsonAndJohnson
- kb:Roche kb:buys kb:AkzoNobel
- kb:JohnsonAndJohnson kb:buys kb:Roche
- kb:JohnsonAndJohnson kb:buys kb:AkzoNobel
- kb:AkzoNobel kb:buys kb:JohnsonAndJohnson
- kb:AkzoNobel kb:buys kb:Roche

This step results in several patterns which are all constructed using OWL individuals, which all have one or more lexical representations. The next step is to substitute the OWL individuals for their different lexical representations. In our case we assume that the companies have only one lexical representation, and that the kb:buys relation has two lexical representations: “buys”, and “acquires”. This step will result in the following lexical representations of the buy-pattern:

- Roche buys Johnson & Johnson
- Roche acquires Johnson & Johnson
- Roche buys Akzo Nobel

- Roche acquires Akzo Nobel
- Johnson & Johnson buys Roche
- Johnson & Johnson acquires Roche
- Johnson & Johnson buys Akzo Nobel
- Johnson & Johnson acquires Akzo Nobel
- Akzo Nobel buys Johnson & Johnson
- Akzo Nobel acquires Johnson & Johnson
- Akzo Nobel buys Roche
- Akzo Nobel acquires Roche

All these lexical representations can then be used in mining the textual items for the occurrence of the pattern. If one occurrence of a pattern is found, it will be interpreted as an indicator of a possible change in the real world. However, one indicator is not a very good basis on which an ontology should be changed. Therefore we will construct a mechanism that gathers several different occurrences of the same pattern. These different occurrences should be from different sources, for example different news feeds, and in a specific time span. If, for example, different news feeds have news items in which the buy pattern for Roche and Akzo Nobel occurred, then this is a very strong indicator that the real world has changed. Once several occurrences have been found we can be more certain about the event. Based on these occurrences the changes can be shown to a domain expert for validation, because it is obviously not desirable to create an inaccurate ontology. The domain engineer will only have to say if a certain event did or did not occur (e.g. “Did Roche buy Akzo Nobel?”), based on the news items in which the event was found. If the domain expert confirms the event, our framework automatically executes the appropriate actions. After the execution of the actions the ontology is updated, thereby reflecting the changed reality.

3.3 Actions

The rule-based ontology learning framework which we propose uses actions to facilitate the actual ontology learning. One or more actions are associated with a pattern to form a rule, which can be executed once the pattern is found. The goal of these actions is to enable knowledge engineers and domain experts to express their knowledge in a simple yet expressive way by combining actions with patterns. To be able to make use of these actions a syntax must be formulated in which the actions can be stored, and based on which the actions can be enforced. In order to provide such a syntax we must have a clear idea of how the actions are going to be used.

In this section we describe a few different rules, for each rule we describe what kind of actions we would like to formulate. Based on these use cases a generalization is performed in order to be able to define which common types of actions are required. Finally, we will evaluate different rule syntaxes which can be used to implement the actions. Based on this evaluation an informed decision is made with respect to the syntax of actions.

3.3.1 Use Cases

In order to formulate a syntax for these actions we have looked at various use cases.

Use Case #1: competitor rule

This rule is fired when a company (subject) is going to compete with a new (or existing) company (object). The required actions are:

1. add the new company (if it does not already exist)
2. create a *hasCompetitor* property instance between the two companies.

Use Case #2: CEO rule

This rule is fired when a company (subject) gets a new CEO (object). The required actions are:

1. add the new CEO (if it does not already exist)
2. remove the old *hasCEO* property instance
3. create a new *hasCEO* property instance to the new CEO

Use Case #3: buy rule

The buy rule is fired when one company (subject) buys another (new) company (object). The required actions are:

1. add the new company (if it does not already exist)
2. remove the *hasCompetitor* property instance between the two companies (if there is one)
3. create a *hasBought* property instance between the two companies
4. remove any *competesWith* property instances between products of the companies

3.3.2 Generalizations

We can distinguish between two kinds of actions: *add actions* and *remove actions* (kind). Both of these actions either apply to an *OWL individual* or an *OWL property* (target). This basically means that there are four different types of actions:

1. add individual
2. add property instance
3. remove individual
4. remove property instance

The adding of new individuals is done automatically when the pattern of a rule is found. If for example the following pattern is found:

[known company] competes with [unknown company]

The *unknown company* is added automatically to the ontology, because of the context in which it appears (e.g. it is competing with a known company).

In order to determine which syntax is most appropriate for these actions we will now evaluate a few alternatives. Based on this evaluation we can make an informed decision.

3.3.3 Alternatives

SWRL

SWRL (Semantic Web Rule Language) [13] is a rule language based on a combination of both OWL, and RuleML [53, 54]. Rules in SWRL are an implication between an antecedent (body), and a consequent (head). The rules can be read as follows: if the conditions in the body hold, then the conditions specified in the head must also hold.

Using SWRL a rule can be formulated which describes in which case two companies (?x1 and ?x2) are competing with each other. In our knowledge base this is the case when these companies manufacture products which have the same target area. A SWRL rule which reflects this knowledge can be found in Figure 3.3.3.

```
Company(?x1) ∧ Product(?y1) ∧ hasManufacturedProduct(?x1, ?y1) ∧
  hasTargetArea(?y1, ?z) ∧ Company(?x2) ∧ Product(?y2) ∧
  hasManufacturedProduct(?x2, ?y2) ∧ hasTargetArea(?y2, ?z) →
  isCompetitorOf(?x1, ?x2)
```

Figure 3.1: Competitor SWRL rule.

This rule accurately matches all cases in which companies compete with one another (i.e. a recall of 100%). However, this rule has two problems. The first problem is the fact that it also matches the case in which ?x1 is the same company as ?x2 (i.e. if they are the same individual). We can solve this by adding a statement like `differentFrom(?x1, ?x2)`, which results in the SWRL rule as shown in Figure 3.3.3. However, this only works if it is explicitly stated that each company is `differentFrom` every other company (e.g. `differentFrom(Roche,`

AkzoNoble), `differentFrom(Roche, JohnsonAndJohnson)`, etc.). In the case of a knowledge base with 300 companies this will result in $300 * 299 = 89.700$ explicit `differentFrom` property instances. Of course, this is not a scalable solution.

```
Company(?x1) ∧ Product(?y1) ∧ hasManufacturedProduct(?x1, ?y1) ∧
  hasTargetArea(?y1, ?z) ∧ Company(?x2) ∧ Product(?y2) ∧
  hasManufacturedProduct(?x2, ?y2) ∧ hasTargetArea(?y2, ?z) ∧
  differentFrom(?x1, ?x2) → isCompetitorOf(?x1, ?x2)
```

Figure 3.2: Competitor SWRL rule using the `differentFrom` property.

The previous SWRL rules do not take into account the fact that companies might have bought each other, in which case they are no longer competitors. We would like to use negation, to specify that the `hasBought` property between two companies should not be present if companies are competitors, as shown in Figure 3.3.3. However, SWRL does not support negation which is the second problem we have when using SWRL rules.

```
Company(?x1) ∧ Product(?y1) ∧ hasManufacturedProduct(?x1, ?y1) ∧
  hasTargetArea(?y1, ?z) ∧ Company(?x2) ∧ Product(?y2) ∧
  hasManufacturedProduct(?x2, ?y2) ∧ hasTargetArea(?y2, ?z) ∧
  differentFrom(?x1, ?x2) ∧ !hasBought(?x1, ?x2) →
  isCompetitorOf(?x1, ?x2)
```

Figure 3.3: Competitor SWRL rule using negation.

SWRL can be used to add individuals and property instances to an ontology, however retractions are not allowed. So it is impossible to remove individuals or instances of properties from an ontology using SWRL. Negation is also not supported at this moment. Another major drawback is the lack of good tools with proper documentation. SWRL can be used in Protégé together with the Jess [55] plugin, and the Racer [56] or Pellet [57] inference engines [58]. However these tools are not all open source and good documentation on the use of SWRL is often lacking.

Self Defined Syntax

A second alternative is to use a self defined syntax to store (and execute) the actions. This self defined syntax can be based on OWL. By creating separate classes for each type of action it might be possible to store the actions in an ontology.

The advantage of creating a specialized syntax is that this syntax can be fully customized with respect to our wishes. However, all the work of defining a syntax and creating an execution mechanism would have to be done from scratch. Besides that, it is usually better to adhere to existing standards.

Triples

The actions can be stored in triple format, which is highly compatible with existing Semantic Web standards. Each triple consists out of a subject, a predicate, and an object. Triples can be connected to each other to form a path expression which makes the use of triples both flexible and expressive.

Storing actions in the form of triples facilitates the use of expressive path expressions which enables the use of both simple (single triple) and more complex (multiple triples, path expression) actions. Using this approach an action become a sequence of triples. These triples can then be used in SPARQL queries, because a SPARQL query essentially consists out of triples. For example action 2 and 3 from the previously stated CEO rule would look like the code in Figures 3.3.3 and 3.3.3 respectively.

```
REMOVE <subject> hasCEO ?x
```

Figure 3.4: Simple SPARQL remove query.

```
CONSTRUCT <subject> hasCEO <object>
```

Figure 3.5: Simple SPARQL add query.

In order to create more expressive path expressions a WHERE clause can

be used, such as shown in the code snippet in figure 3.3.3 which depicts action 4 of the buy rule.

```
REMOVE ?x competesWith ?y
WHERE
{
    <subject> hasProduct ?x
    <object> hasProduct ?y
}
```

Figure 3.6: Advanced SPARQL remove query.

SPARQL [59] can be used for implementing the actions using the convenient triple format. SPARQL supports the use of a CONSTRUCT clause, and a WHERE clause, together with the use of the triple format. Because of this SPARQL is ideal for implementing the “add actions” (i.e. add individuals and add property instances). The problem with a SPARQL implementation lies in the fact that at this moment SPARQL does not support removal operations. This means that it is impossible to use SPARQL for the removal of individuals and properties. There are plans for adding this functionality in the future [60], however these changes are not fully implemented yet.

To enable novice users to create actions in triple format an interface has to be build, that allows them to create a CONSTRUCT, a REMOVE and a WHERE clause. Our previous work on SPARQLViz [61] can be of use for this purpose.

3.3.4 Conclusion

In this section we have evaluated three different alternatives for storing and executing actions. SWRL looks like a good candidate, however the lack of good tools is a problem. Besides this it can not be used for retraction and it does not facilitate negation. Because of these drawback it is not useful for us.

Another alternative is defining our own syntax. This gives us the freedom to fully customize both the storing and execution mechanisms. However, doing so might be a tedious job. Besides that it is better to adhere to existing standards

in order to be able to easily re-use this framework in another context.

The use of triple format to store the actions, and SPARQL to execute these actions makes maximum use of the already existing technologies. The main problem is that SPARQL currently does not officially supports removal (and update) operations. However, there is a syntax defined for these operations [60], which is currently being implemented in Jena [14] / ARQ [21]. This implementation is still in beta status, but it can be used. An alternative for this would be to store all the actions in triple format, and split the execution in two separate environments. The add actions will be executed with SPARQL. The remove actions will be executed by an execution engine which we can write ourselves.

Based on this research we conclude that it is best to use triple format to store the actions. All add actions can be executed with the use of SPARQL. The remove actions can be executed through SPARQL with the use of beta software.

Chapter 4

Rule Engine

Hell, there are no rules here– we're trying to accomplish something.

– Thomas A. Edison, 1847 - 1931

In this chapter our rule-engine is discussed. Section 4.1 provides a brief introduction, after which the rule editor is discussed in section 4.2, followed by a description of the event detector in section 4.3. The procedure of executing the actions is discussed in section 4.5.

4.1 Introduction

The rule engine is used in four successive phases to discover and process events based on news items. These phases are:

1. Mine text items for patterns
2. Create event if a pattern is found
3. Determine validity of event through domain expert
4. Execute appropriate actions if an event is valid

The rule engine consists of a rule editor (described in section 4.2), an event detector (see section 4.3), a validation environment (described in section 4.4), and an action execution engine (discussed in section 4.5).

The editor can be used to create patterns, actions and the relations which are used by patterns. These relations are instances of a special class we call “Relation”. Using the editor the domain experts can construct the semantic rules. The event detector is used for mining text items (in our case news items from RSS feeds) for the occurrence of the patterns of the semantic rules (i.e., the event detector mines text items for occurrences of events). Using the validation environment domain experts are able to determine if the found events are valid. They can also modify the events in case the event detector made an error. If an event is validated the action execution engine is used to execute the actions, associated with the rule which is used for finding the event.

4.2 Rule Editor

We have created a rule editor which allows knowledge engineers to construct the semantic rules which are used in the learning process. Using this editor relations can be created and patterns can be formulated, based on the ontology that is used in the learning process. Figure 4.2 shows the user interface which is used when the domain experts want to add or modify relations. Using this interface relations and their lexical representations (synonyms) can be created and modified. Each relation is an individual of the class “Relation” in our knowledge base. This class contains all the relations for which the domain expert defined semantic rules.

Figure 4.2 shows the user interface which is used to create and modify the semantic rules. Through this interface the pattern of the rule can be created by choosing a subject, a relation, and an object. In this figure the previously discussed buy-rule is depicted.

We have also implemented an editor for the actions which are associated with the rules. Figure 4.2 shows the user interface which is used to create and modify these actions. In this figure action 4 of the buy-rule (remove any *competesWith* property instances between products of the companies) is depicted. By action editor allows the user to specify what kind of action he wants to create (add or

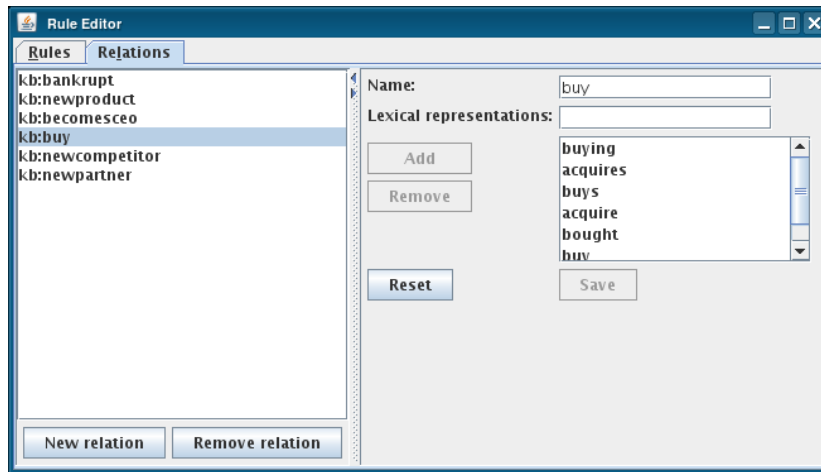


Figure 4.1: The rule editor, showing the interface for editing relations.

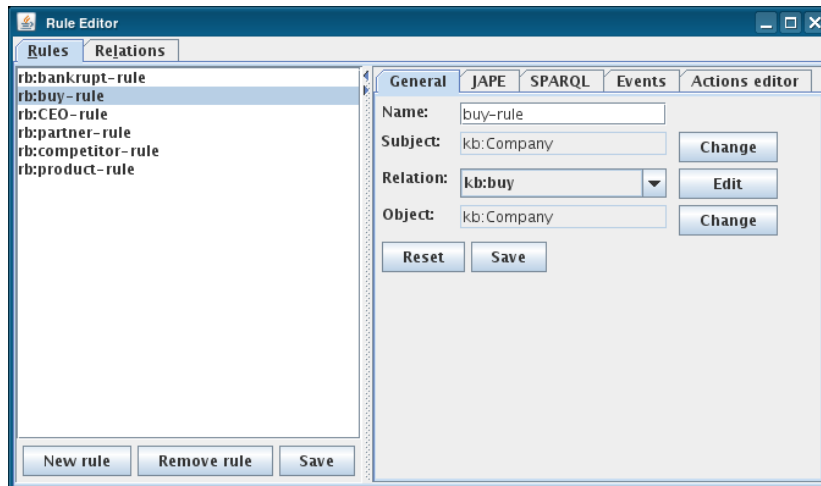


Figure 4.2: The rule editor, showing the interface for editing rules.

delete). After that the main clause and the WHERE clause of the action can be created. These clauses consist out of triples which can be created, altered, and removed using the action editor.

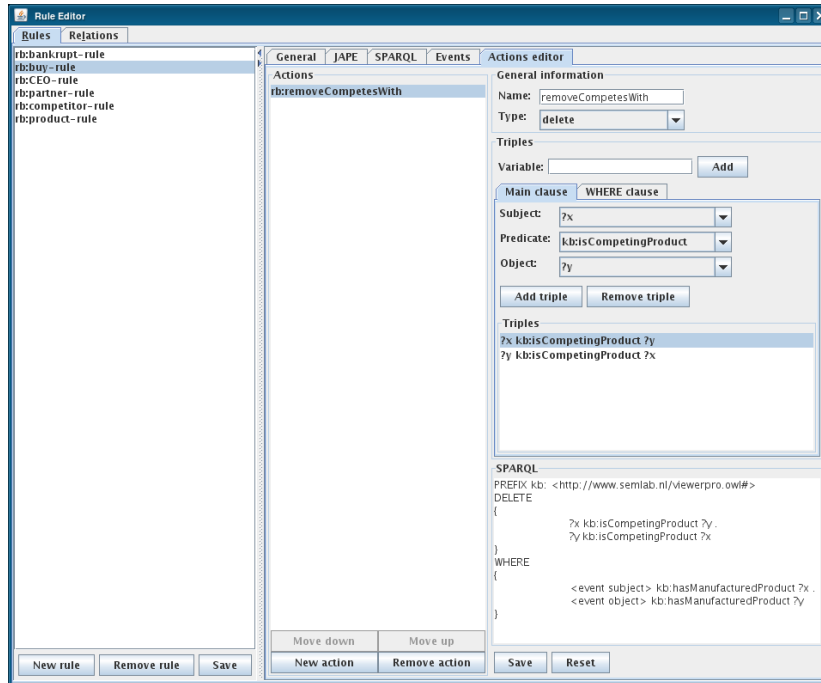


Figure 4.3: The action editor.

4.3 Event Detector

4.3.1 Introduction

The actual pattern matching is done using GATE (General Architecture for Text Engineering), which is an environment supporting the research and development of language processing software [24]. GATE provides its users with numerous components (called resources), which can be used to construct a language processing application. It also allows its users to develop their own modules or extend the existing ones.

In order to search the news items for occurrences of events, we mine all the news items in the RSS feeds which have not yet been processed by our event detector. If a pattern associated with a rule is found in the text, we create an event if it does not already exist. If multiple instances of the same event are found (i.e., multiple news items contain the same pattern), these news items are

added to this event. For example we might find the following three news items (italic texts denote companies identified by a domain expert, bold texts denote relations identified by a domain expert):

1. Pharmaceuticals giant *AstraZeneca* **buys** *MedImmune* for \$15.6 billion
2. *QIAGEN* Signs Agreement for the **Acquisition of** *eGene*
3. *AstraZeneca* to **buy** *MedImmune*

We parse these news items with respect to the following pattern (which is associated with the buy-rule):

[Company] buys [Company]

The pattern matching process first identifies all relevant concepts in a news item, with respect to the subject and the object of a pattern. In the case of the previous three news items and the buy-pattern this means that all the relevant companies are found, based on their lexical representations stored in the ontology. The news items are also searched for the occurrence of unknown companies, which are not yet stored in the ontology. This is done using the ANNIE Gazetteer, the ANNIE Named Entity Transducer, and the ANNIE Ortho Matcher (all from GATE), more information about these GATE components can be found in subsection 4.3.2. Using these GATE components our application automatically identifies unknown companies, thereby learning at the instance level of the ontology.

After that, the event detector looks for the presence of the relevant relation. In the case of the buy-pattern the event detector tries to locate a lexical representation of the buy-relation in the news item. If both a subject, and an object together with a relation are found in the news item a new event is created, if this does not already exist. If the event already exists, the news item is added to the event.

Based on the buy-pattern our software identifies two events, the italic texts denote lexical representations of companies as found by the event detector (the non-italic text denotes a company that is not recognized by the event detector):

1. *AstraZeneca* $\xrightarrow{\text{buys}}$ *MedImmune* (with news items #1 and #3)
2. *QIAGEN* $\xrightarrow{\text{buys}}$ *eGene* (with news items #2)

The first event is a *full match*, because we have identified both a subject and an object. The second event is a *partial match*, because we were only able to identify a subject (the company *QIAGEN*), and the buy-relation (based on its lexical representation “*Acquisition of*”). The buy-pattern requires both a subject and an object to be valid, so this event can not be validated by a domain expert at this point, because the object is missing. However, when a domain expert reviews the event, he might recognize *eGene* as being a company based on both his own knowledge, and the context of the news item in which *eGene* appears. Based on this event (and the corresponding news item) the domain expert can add *eGene* to the ontology, after which this event has both a subject (*QIAGEN*) and an object (*eGene*). Now that the event has both a subject and an object it is a full match, so it can be validated. In the ideal case *eGene* would be recognized by GATE automatically as being an unknown company, however at this point we are not capable of recognizing all unknown companies automatically. GATE recognizes companies based on Gazetteer lists, and JAPE rules. These JAPE rules look in the context of the text. If, for example, a company is found in a sum up with other companies GATE is capable of recognizing the unknown company as being a company. Take the following sentence for example: “Larry Page works for Google, Microsoft, and eGene.”. Based on this sentence GATE is able to deduce that “eGene” is a company, because of the context in which it appeared. In the short RSS news items GATE is not always capable of identifying unknown companies because of the lack of context.

In the following subsection we give a brief overview of our GATE pipeline, which is used to identify the patterns in the news items.

4.3.2 GATE Pipeline

In order to match the pattern of a rule in a news item we constructed a GATE pipeline. Each news item will be pushed through this pipeline in order to check if one of the lexical representations of the lexico-syntactic pattern of a rule can be matched.

This subsection discusses the components that together comprise the GATE pipeline in the order in which they are used. Some components we use are distributed with GATE by default, these are: Document Reset, ANNIE English Tokeniser, ANNIE Gazetteer, ANNIE Sentence Splitter, ANNIE Part-Of-Speech (POS) Tagger, ANNIE Named Entity (NE) Transducer, and the ANNIE Ortho Matcher. Besides those components we use a third party component: Apolda [62] (Automated Processing of Ontologies with Lexical Denotations for Annotation). The last three components of the pipeline are JAPE Transducers which we have created ourselves: Pre-pre-processing JAPE Transducer, Pre-processing JAPE Transducer, and the Pattern Matching JAPE Transducer. During execution of the event detector the last two JAPE Transducer (which are actually JAPE grammars) are generated on the fly by the detector, based on the rule that is currently being matched. The first JAPE Transducer does some general pre-pre processing, so it is constant.

Document Reset The Document Reset component resets the document to its original state by removing all the annotation sets and the corresponding annotations. An annotation set contains multiple annotations, each annotation has multiple instances. An example of an annotation is “Sentence”, which has all sentences in a text as its instances. Another example is “Organization”, which has as its instances all organizations in the text identified by GATE. An annotation set is used to group several annotations into one set. These annotation sets can be used to separate annotations which are not related.

The Document Reset is used at the beginning of the pipeline to remove all old annotations.

ANNIE English Tokeniser This component splits the text into simple tokens such as punctuation, spaces, numbers, and words of different types (for example: words in uppercase and lowercase). The English Tokeniser consists of a normal tokeniser, and a JAPE transducer. This transducer is used to concatenate various tokens to create constructs like “50’s”, and “don’t”.

We make use of the alternate rule set for the ANNIE English Tokeniser, which is also distributed with GATE. When using this alternate rule set the tokeniser recognizes hyphenated words better. This means that words like “Sanofi-Aventis” are annotated as one token.

Because of the tokenization more advanced analysis, such as done by Apolda, can be performed in the following components.

ANNIE Gazetteer The ANNIE Gazetteer is used in order to classify certain types of words, for example: cities, organizations, countries, dates, etc. The gazetteer contains lists of words for which a major type, and an optional minor type are defined. If words on this list are identified in the text, they are annotated with the major-, and minor type. The minor type is used to provide more detailed information about the annotation. For example: a company is a specific type of organization. Below is a small section of the list for companies called `company.lst` with the major type `organization`, and minor type `company`:

RCA

Reader’s Digest

Reebok

Reed

Reed Elsevier

If one of these pieces of text is found it will get annotated with a major type `organization`, and minor type `company`. Once the text is annotated using the gazetteer the JAPE grammars can match all companies by specifying the minor type `company`. All organizations can be matched by specifying the more general major type `organization`.

ANNIE Sentence Splitter The Sentence Splitter is used to split the text into sentences, which is a requirement of the POS-tagger.

ANNIE Part-Of-Speech (POS) Tagger The Part-Of-Speech tagger [63] of GATE is a modified version of the Brill tagger [64]. It provides POS-tags as an annotation for every word or symbol in the text. POS-tags can be used to identify verbs, singular nouns, plural nouns, adjectives, etc.

ANNIE Named Entity (NE) Transducer The Named Entity Transducer (also known as a Semantic Tagger) makes use of annotations assigned in earlier phases. Based on this annotations named entities can be found and annotated. For example: the two successive words “Larry”, and “Page” are annotated together thereby creating the named entity “Larry Page” with is annotated as type Person. Using the NE transducer new organizations, persons, locations, and dates can be found. Companies are a specific type of organization, so they can also be found.

ANNIE Ortho Matcher The Ortho Matcher (also known as the Orthographic Coreference component or NameMatcher) is used to add identity relations between named entities found by the Named Entity Transducer. Using the Ortho Matcher strings which refer to the same thing can be found, e.g. “IBM” and “Big Blue” both refer to the company IBM.

Apolda (Automated Processing of Ontologies with Lexical Denotations for Annotation) Apolda [62] is a component which has similarities with the ANNIE Gazetteer, the biggest difference lies in the fact that it uses terms from an ontology to classify words instead of gazetteer lists. Apolda searches the text for occurrences of OWL annotation properties of the classes and instances of an ontology. The found matches are annotated with the name of the OWL instance (or class) against which the piece of text is matched. Using this component text can be mined for the occurrences of terms from an OWL ontology.

JAPE Transducer - Pre-pre-processing After annotating the text with Apolda each term in the text that corresponds to an OWL class or OWL individual is annotated with that class or individual. However, in our matching process we only want to use OWL individuals, therefore this JAPE transducer removes all class annotations from the annotation set.

JAPE Transducer - Pre-processing Each rule has a lexico-syntactic pattern (which has multiple lexical representations) associated with it which is used to identify events in text. In order to be able to find the lexical representations of a lexico-syntactic pattern this pattern is dynamically converted to two JAPE grammars each time a GATE pipeline is being executed. These two JAPE grammars are executed successively in two separate phases. The grammar in the first phase (the pre-processing phase) is used to copy all relevant annotations to a temporary annotation set. By using a temporary annotation set for the relevant annotations the next phase automatically discards the non-relevant annotations, because these non-relevant annotations are not copied to the temporary annotation set. Only things that appear in the pattern of the rule are relevant, so for the buy-rule (`[kb:Company] kb:buys [kb:Company]`) only company annotations, and annotations which reflect the buy-relation are copied to the temporary annotation set.

JAPE Transducer - Pattern Matching The second JAPE grammar (the pattern matching phase) is used to match the pattern, based on the annotations in the temporary annotation set created in the previous phase. In this pattern matching phase the JAPE grammar is used to identify full or partial matches of the lexico-syntactic pattern in the relevant annotations.

4.4 Event Validation

Identified events are stored in an ontology, together with the news items in which they are found. In the validation environment the domain expert can review all the identified events. If an event is a full match (which means both

a subject and an object -if this is required- are found), the domain expert can validate the event. Once an event is validated the associated actions are executed, thereby modifying the ontology based on the identified event. Figure 4.4 gives an overview of the validation environment. Colored rows indicate events which are not yet validated. The italic subjects and objects are concepts which are unknown at this point, they are discovered in the news items by the GATE pipeline. If an event containing such unknown concepts is validated, these concepts are added to the knowledge base.

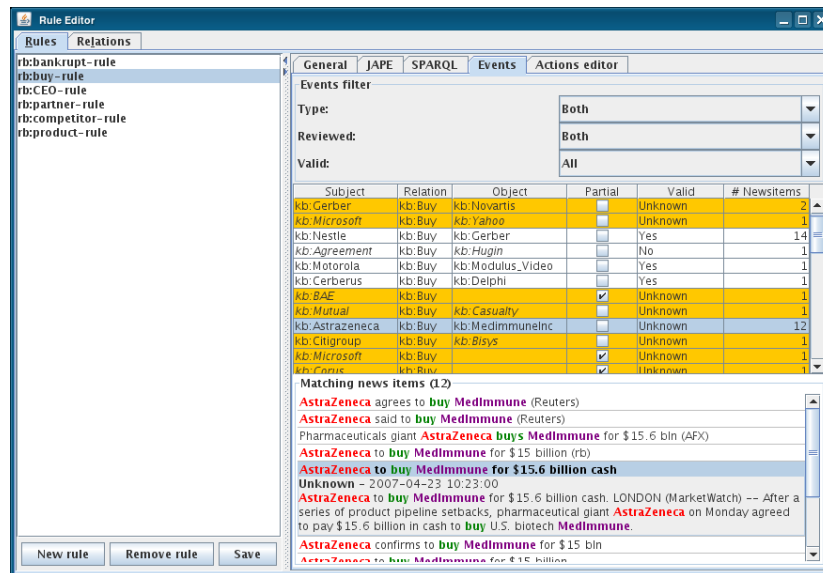


Figure 4.4: The validation environment of the rule engine.

4.5 Action Execution Engine

If a domain expert validates an event the appropriate actions should be executed. This is done in the following steps:

1. Retrieve the appropriate actions
2. Substitute the <subject> in the actions for the actual subject of the event

3. Substitute the `<object>` in the actions for the actual object of the event (if there is one)
4. Execute the actions using a SPARQL engine

The actions associated with the rule whose pattern is matched by the event are first retrieved from the rule base. An example of such an action can be found in Figure 4.5. This action is associated with the buy-rule, it removes the *competesWith* property instance between two products whose producers do no longer compete with each other (because one bought the other).

```
REMOVE ?x competesWith ?y
WHERE
{
    <subject> hasProduct ?x
    <object> hasProduct ?y
}
```

Figure 4.5: Abstract action before substitution.

As can be seen from the previous figure this abstract action can not be executed directly. Before the action is ready for execution the `<subject>` and `<object>` must be substituted for the actual subject and object as found in the event. If for example *Roche* buys *Akzo Nobel*, the action will look like Figure 4.6 after the substitutions. This concrete action can then be executed using an experimental implementation of ARQ [21] which supports SPARQL-U [60] which is still under development.

```
REMOVE ?x competesWith ?y
WHERE
{
    kb:Roche hasProduct ?x
    kb:AkzoNobel hasProduct ?y
}
```

Figure 4.6: Action after substitution.

Chapter 5

Evaluation

True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information.

– Winston Churchill, 1874 - 1965

The subject of this chapter is the evaluation of the results of the learning process. To validate the results we compare the events identified by the rule-engine with those identified by a domain expert, based on the same set of news items.

5.1 Introduction

Performance evaluation of the ontology learning process is a problem in this field of research, because there are no predefined measures that everybody relies on [46]. We decided to use the following measures: accuracy (A), error (E), precision (P), recall (R), and F_1 . We chose these measures, because they are well-known and relevant statistical measures. Moreover we have defined our own measure for usefulness (U). Using this measures we evaluate the result of the rule-engine against the reference standard, as defined by an expert.

It should be noted that the quality of the event detector for a large part relies on the input that it gets. This means that without a high quality knowledge base, which contains the proper lexical representations for both the concepts

and the relations, the amount of successfully identified events will dramatically decrease. Besides that, the clarity of the parsed text is also important. If the text which is being mined is very unstructured, it is hard to match patterns. When using an adequate knowledge base and well formulated text, we expect that the event detector performs fairly well.

In section 5.2 we describe the experiment which we conducted and discuss the several runs and the reference standard created by a domain expert. This is followed by a discussion of the used measures in section 5.3. In section 5.4 we describe the results with respect to the used measures. Based on the runned experiment section 5.5 presents our conclusions.

5.2 Experimental Setup

5.2.1 Introduction

In order to be able to evaluate the output of the rule-engine we have done an experiment with 1.736 news items. We retrieved these news items from the following RSS feeds: <http://www.marketwatch.com>, <http://uk.finance.yahoo.com>, and <http://feeds.reuters.com/reuters/businessNews>. Besides that, we have received a data set of news items from Reuters via Citigroup. All these news items are aggregated to one data set which we used in our experiment. We call this data set of news items *NI*. Our rule-engine processed this data set in order to find events associated with the previously discussed buy-rule (i.e., buy-events). We choose to use the buy-event in the evaluation process because it is the most frequently encountered event in our data set.

5.2.2 First Run - Strict Pattern

We did three processing runs with our rule-engine. In the first run the rule-engine strictly looked for the pattern `[kb:Company] kb:buys [kb:Company]`. Which means that the rule-engine only looks for known companies (i.e., companies which are in the knowledge base), and tokens which are explicitly annotated by GATE as being a company. We call the set of identified events from this run

E_{DS} , because the events are identified by the event detector (D) with the strict (S) pattern.

5.2.3 Second Run - Relaxed Pattern

For the second run of the rule-engine we relaxed the restrictions on the pattern so it also includes tokens which are annotated by GATE as being an organization (not specifically a company) or which are annotated as Unknown. GATE annotates tokens as being Unknown in case it knows the token is something special (for example: if it starts with a capital letter, or if it is written entirely uppercase), but it does not know its specific type. We call the set of identified events from this run E_{DR} , because they are associated with the more relaxed (R) buy-pattern.

5.2.4 Third Run - Hybrid Pattern

The third run of the rule-engine combined both previous approaches. It uses the relaxed pattern of the second run with the extra condition that either the subject or the object of the found event is a known company (i.e., the company should be in the knowledge base). This extra condition on either the subject or the object combines the relaxed pattern with a more strict approach. We call the set of identified events from this run E_{DH} , because the events are identified by the event detector with the hybrid (H) pattern.

5.2.5 Reference Standard

A reference standard is needed against which the results of the rule-engine can be evaluated. This reference standard (E_{RS}) is a set of events created by manually identifying buy-events in the data set. Each buy-event (for example “Nestle buys Gerber”) is supported by one or more news items which denote this event.

To summarize: each event is a set of one or more news items, the reference standard is a set of all buy-events manually identified by a domain expert. The mathematical definition of the reference standard can be found in equation (5.1),

where E_{ALL} denotes a set which contains all valid events in the data set.

$$E_{RS} = \{e \in E_{ALL} \mid e \text{ denotes a buy-event}\} \quad (5.1)$$

Basically the reference standard is a set of events, these events are sets of news items denoting the same event instance, as can be seen from equation (5.2).

$$E_{RS} = \{e_1\{n_1, n_2, \dots, n_n\}, e_2\{n_1, n_2, \dots, n_n\} \dots\} \quad (5.2)$$

5.3 Measures

5.3.1 Introduction

In this section we discuss the different measures which we have computed with respect to the output of the rule-engine. We have computed these measures for all three runs, the results of these computations can be found in section 5.4. Please note that we calculate the measures on news item level, because the event-detector takes a decision for each news item (i.e. “does this news item denote an event, and if so, which one?”). This means that, for each calculation, we use the amount of news items in the event sets. The symbols we use in the calculations are explained in table 5.1.

Symbol	Meaning
E_D	Events identified by the event detector of the rule engine, E_{DS} , E_{DR} , or E_{DH} .
E_{RS}	Events identified by the domain expert (the reference standard).
E_{CI}	Events correctly identified by the event detector; $E_D \cap E_{RS}$.
E_U	Events identified by the event detector which are useful; $E_U = E_{CI} \cup \{e \in E_D \mid e \text{ is partially valid}\}$.
E_T	All identified events, by either the event detector or the domain expert; $E_D \cup E_{RS}$.
NI	All news items in the data set.

Table 5.1: The symbols used in the measures.

Table 5.2 contains the contingency table which shows the relationships between the used symbols. The first column of the data in the contingency table

reflects the events (and thus the news items) that were identified by the domain expert. The second column reflects the events (news items) which are not identified by the domain expert, this means that these news items do not denote a buy-event. The first row of the data in the table reflects the events (news items) which are identified by the event detector as being buy-events. The second row reflects the events (news items) that are not identified by the event detector as being an event.

Detector	Expert	
	Yes	No
Yes	E_{CI}	$E_D - E_{RS}$
No	$E_{RS} - E_D$	$NI - E_T$

Table 5.2: The contingency table.

5.3.2 Accuracy

The accuracy measure is used to determine the portion of news items that are correctly identified as either belonging to an event or not. It is defined as the sum of the number of news item belonging to the events correctly identified by the event detector, and the number of news items belonging to the events the event detector correctly ignored. This sum is divided by the total amount of news items in the data set. It falls in the range from 0 to 1, where 1 is the best score. The exact definition of this measure can be found in equation (5.3).

$$\begin{aligned}
 A &= \frac{|E_{CI} \cup (NI - E_T)|}{|NI|} \\
 &= \frac{|(E_D \cap E_{RS}) \cup (NI - (E_D \cup E_{RS}))|}{|NI|} \\
 &= \frac{|NI - (E_D \Delta E_{RS})|}{|NI|}
 \end{aligned} \tag{5.3}$$

Most of the calculations in this and the following sections are pretty straightforward. The most complicated one is depicted in equation (5.4). It shows the use of an operator called the symmetric difference.

$$E_D \Delta E_{RS} \tag{5.4}$$

In normal circumstances this is a basic operation which results in a set which contains all elements which are either present in E_D or E_{RS} , but not in both. The elements that appear in both sets are the events (and thus the news items), which are correctly identified by the event detector, i.e. E_{CI} . This means that the set which results from the symmetric difference operation contains all the news items for which the event detector made an error, either by classifying it incorrectly or ignoring the news item when it should not do so.

In our situation with sets of events which are denoted by news items we have to take duplicate news items into account. It might be the case that a news item is detected by the event detector as belonging to an event, but the event itself is invalid. This can be the case if, for example, the detected event is partial. This means that the news item is not correctly identified, therefore it is present in E_D but not in E_{CI} . This same news item can also be present in E_{RS} , because the domain expert has identified the news item as belonging to a specific event. In this case the same news item is associated with two different events which are in E_D and E_{RS} , but not in E_{CI} . When using the symmetric difference operator this news item appears twice in the resulting set. However, we only want to count the amount of unique news items for which the event detector made an error. Therefore we have removed all the duplicates from the resulting set of news items.

5.3.3 Error

The error measure is used to determine the portion of all news items in the data set that are incorrectly classified by the event detector. It is defined by the sum of the number of news items which are incorrectly classified by the event detector, and the number of news items the event detector failed to classify. This sum is divided by the total amount of news items in the data set, in order to determine the portion of news items for which the event detector made an error. It falls in the range from 0 to 1, where 0 is the best score. Together accuracy and error are 1. The exact definition of this measure can be found in

equation (5.5).

$$E = \frac{|(E_D - E_{RS}) \cup (E_{RS} - E_D)|}{|NI|} = \frac{|E_D \Delta E_{RS}|}{|NI|} \quad (5.5)$$

5.3.4 Precision

The precision measure is used to determine how many of the news items classified by the event detector are actually correct. It falls in the range from 0 to 1, where 1 is the best score. The exact definition of this measure can be found in equation (5.6).

$$P = \frac{|E_{CI}|}{|E_D|} \quad (5.6)$$

5.3.5 Recall

The recall measure is used to determine the portion of news items in the data set that are correctly classified by the event detector. It falls in the range from 0 to 1, where 1 is the best score. The exact definition of this measure can be found in equation (5.7).

$$R = \frac{|E_{CI}|}{|E_{RS}|} \quad (5.7)$$

5.3.6 F_1

There is a clear trade off between precision and recall, therefore the F_1 measure is needed. The F_1 measure is used to compute an even combination of precision and recall. It falls in the range from 0 to 1, where 1 is the best score. When improving the event detector the F_1 measure is the measure that should be maximized, because it takes both precision and recall into account. The exact definition of the F_1 measure is shown in equation (5.8).

$$F_1 = \frac{2 * P * R}{P + R} \quad (5.8)$$

5.3.7 Usefulness

We have defined our own measure which denotes the degree of usefulness of each run. It is determined by the amount of news items, belonging to events identified

by the event detector which are useful (E_U), divided by the amount of news items belonging to all the events identified by the event detector. Useful events can be valid or partially valid, the partially valid events are quite useful, because the domain expert can complete them and make them valid with a minimum amount of effort. The only thing that has to be done is to identify the missing valid subject or object. A definition of E_U can be found in equation (5.9).

$$E_U = E_{CI} \cup \{e \in E_D \mid e \text{ is partially valid}\} \quad (5.9)$$

This measure falls in the range of 0 to 1, where 1 is the best score. The exact definition of this measure can be found in equation (5.10).

$$U = \frac{|E_U|}{|E_D|} \quad (5.10)$$

5.4 Results

In this section we outline the results for each of the previously discussed measures. We first discuss the measures with respect to the run with the strict pattern. After that the run with the more relaxed pattern is discussed. Finally, the measurements for the run with the hybrid pattern are outlined.

5.4.1 First Run - Strict Pattern

The data resulting from the run with the strict pattern is shown in Table 5.3. In this table the column “# events” contains the number of unique event instances (e.g. “Gerber buys Nestle”, etc.) in the given sets. The column “# news items” contains the number of unique news items which support the events.

	# events	# news items
Event detector (E_{DS})	37	78
Domain expert (E_{RS})	20	75
Correctly identified ($E_{CI} \Leftrightarrow E_{DS} \cap E_{RS}$)	4	17
Useful (E_U)	20	57

Table 5.3: The data when running the rule-engine using the strict pattern.

Based on this data we calculate the previously discussed measures on the news item level.

Accuracy

$$A = \frac{|NI - (E_{DS} \triangle E_{RS})|}{|NI|} = \frac{1,736 - 81}{1,736} = 0.95 \quad (5.11)$$

Error

$$E = \frac{|E_{DS} \triangle E_{RS}|}{|NI|} = \frac{81}{1,736} = 0.05 \quad (5.12)$$

Precision

$$P = \frac{|E_{CI}|}{|E_{DS}|} = \frac{17}{78} = 0.22 \quad (5.13)$$

Recall

$$R = \frac{|E_{CI}|}{|E_{RS}|} = \frac{17}{75} = 0.23 \quad (5.14)$$

F_1

$$F_1 = \frac{2 * P * R}{P + R} = \frac{2 * 0.22 * 0.23}{0.22 + 0.23} = \frac{0.1012}{0.45} = 0.22 \quad (5.15)$$

Usefulness

$$U = \frac{|E_U|}{|E_{DS}|} = \frac{57}{78} = 0.73 \quad (5.16)$$

5.4.2 Second Run - Relaxed Pattern

The data resulting from the run with the relaxed pattern is shown in Table 5.4.

	# events	# news items
Event detector (E_{DR})	62	99
Domain expert (E_{RS})	20	75
Correctly identified ($E_{CI} \Leftrightarrow E_{DR} \cap E_{RS}$)	10	38
Useful (E_U)	25	58

Table 5.4: The data when running the rule-engine using the relaxed pattern.

Based on this data we calculate the previously discussed measures on the news item level.

Accuracy

$$A = \frac{|NI - (E_{DR} \triangle E_{RS})|}{|NI|} = \frac{1,736 - 80}{1,736} = 0.95 \quad (5.17)$$

Error

$$E = \frac{|E_{DR} \triangle E_{RS}|}{|NI|} = \frac{80}{1,736} = 0.05 \quad (5.18)$$

Precision

$$P = \frac{|E_{CI}|}{|E_{DR}|} = \frac{38}{99} = 0.38 \quad (5.19)$$

Recall

$$R = \frac{|E_{CI}|}{|E_{RS}|} = \frac{38}{75} = 0.51 \quad (5.20)$$

F_1

$$F_1 = \frac{2 * P * R}{P + R} = \frac{2 * 0.38 * 0.51}{0.38 + 0.51} = \frac{0.3876}{0.89} = 0.44 \quad (5.21)$$

Usefulness

$$U = \frac{|E_U|}{|E_{DR}|} = \frac{58}{99} = 0.59 \quad (5.22)$$

5.4.3 Third Run - Hybrid Pattern

The data resulting from the run with the hybrid pattern is shown in Table 5.5.

	# events	# news items
Event detector (E_{DH})	30	62
Domain expert (E_{RS})	20	75
Correctly identified ($E_{CI} \Leftrightarrow E_{DH} \cap E_{RS}$)	9	36
Useful (E_U)	18	50

Table 5.5: The data when running the rule-engine using the hybrid pattern.

Based on this data we calculate the previously discussed measures on the news item level.

Accuracy

$$A = \frac{|NI - (E_{DH} \triangle E_{RS})|}{|NI|} = \frac{1,736 - 52}{1,736} = 0.97 \quad (5.23)$$

Error

$$E = \frac{|E_{DH} \triangle E_{RS}|}{|NI|} = \frac{52}{1,736} = 0.03 \quad (5.24)$$

Precision

$$P = \frac{|E_{CI}|}{|E_{DH}|} = \frac{36}{62} = 0.58 \quad (5.25)$$

Recall

$$R = \frac{|E_{CI}|}{|E_{RS}|} = \frac{36}{75} = 0.48 \quad (5.26)$$

F_1

$$F_1 = \frac{2 * P * R}{P + R} = \frac{2 * 0.58 * 0.48}{0.58 + 0.48} = \frac{0.5568}{1.06} = 0.53 \quad (5.27)$$

Usefulness

$$U = \frac{|E_U|}{|E_{DH}|} = \frac{50}{62} = 0.81 \quad (5.28)$$

5.5 Conclusion

An overview of the results of the evaluation can be found in Table 5.6. This table shows that the results are significantly better when using the hybrid pattern with the exception of recall, the strict pattern gives the worst results.

The overall accuracy is very good. The accuracy of the strict and relaxed patterns is the same, the accuracy of the hybrid pattern is slightly better. The reason for this is that most of the news items are not associated with a buy event. All runs have successfully ignored a large portion of these news items, this results in an almost equal accuracy.

Measure	Strict pattern	Relaxed pattern	Hybrid pattern
Accuracy	0.95	0.95	0.97
Error	0.05	0.05	0.03
Precision	0.22	0.38	0.58
Recall	0.23	0.51	0.48
F_1	0.22	0.44	0.53
Usefulness	0.73	0.59	0.81

Table 5.6: An overview of the measures for the three runs.

The error rate is lowest when using the hybrid pattern. When using the relaxed pattern the event detector is capable of recognizing more companies, then when it uses the strict pattern. However, this results in some events being out of domain, an example of this is the event “Thomson buys Reuters”. Although such an event is correctly recognized we do not want to use it in our learning framework, because it learns facts which are not in our domain. Therefore we consider these events invalid in our context. When using the hybrid pattern we have the restriction that either the subject or object must be a known company, thereby making sure that the event is always in our domain. This reduces the error rate of the event detector. The error rate is low, especially when using the hybrid pattern.

The precision of the event detector is highest when using the hybrid pattern. The same rationale as for the error rate applies here. The event detector recognizes more companies with the relaxed pattern, thereby being able to identify events which otherwise would not have been recognized or which would have been partial matches (because the subject or object could not be found). However, because of the restriction that either the subject or the object must be present in the knowledge base, we reduce the amount of invalid matches.

The recall of the run with the relaxed pattern is higher than the other two runs. The reason for that is the fact that, when using the relaxed pattern, the event detector is capable of recognizing more events than with the strict pattern. This results in a higher recall for the relaxed run, however the precision suffers. The hybrid pattern also allows for more recognition but, because of the

restriction that either the subject or the object must be in the knowledge base, results in a slightly lower recall. However, the precision of the hybrid pattern is significantly better than the precision of the relaxed pattern.

The F_1 measure is higher when using the hybrid pattern. The F_1 measure is computed based on the precision and the recall, because of this the same logic applies as for those measures. Based on this measure we can clearly see that the run with the hybrid pattern is the most successful.

The usefulness of the identified news items is highest when using the hybrid pattern, followed by the strict pattern. The relaxed pattern scores the lowest on usefulness. The reason for this is that the usage of the relaxed pattern introduces a lot of out of domain events, which we have marked as invalid. The hybrid pattern solves this problem due to its restriction that either the subject or the object of an event must be present in the knowledge base.

As an overall conclusion we can say that the event detector performs fairly well, especially when using the hybrid pattern. Future work should focus on improving the recall, and the precision. However, at this point the event-detector is already useful, because it significantly reduces the amount of news items that domain experts have to review in order to find a buy-event. Once such a buy-event is found and validated, our rule-engine can execute the appropriate actions, thereby updating the ontology.

Chapter 6

Conclusions and Future Research

Properly designed, the Semantic Web can assist the evolution of human knowledge as a whole.

– Tim Berners-Lee, The Semantic Web (17 May 2001)

In this chapter we present our findings. Based on our literature review and experiment we test our earlier defined hypotheses on validity in section 6.1. In section 6.2 we discuss the areas for future research.

6.1 Conclusions

In the beginning of this thesis we defined the following research question:

How to exploit domain knowledge in a pattern-action based approach for ontology learning?

To answer this question we defined three different hypotheses:

1. Expert knowledge is needed for facilitating ontology learning.
2. A formal rule syntax is useful when formalizing expert knowledge.
3. By using an expert defined rule base, an ontology learning mechanism can be created.

We will now determine the validity of each of our hypotheses in order to answer the research question.

Our literature survey revealed that a lot of research already showed that expert knowledge is needed to support and validate the ontology learning process. In our own research we found that domain experts are needed to formulate rules, and to validate the events we found in news items. Because we are trying to learn new things, we have to be aware of the risk of learning things that are out of domain, or even worse learning things which are invalid. Because of this we believe that it is safe to say that expert domain knowledge is needed to supervise the learning process.

We found the use of rules very useful when mining news items for specific events. We used the well known pattern based approach from Hearst [33, 9], which we extended with domain specific syntactic arguments, based on the knowledge base. By using these patterns, rules can be formulated which associate patterns with one or more actions in order to update the ontology. By formalizing the syntax for these rules one can easily create and modify rules. We have also built a rule editor which facilitates the easy creation, modification, and removal of rules. Using this editor and the underlying rule syntax, experts can easily define their own rules which are very useful when mining news items for events.

The expert defined rule base is used to mine news items for the occurrence of specific events. Based on these events the ontology is updated to reflect the changed reality. We have found that an expert defined rule base is a very useful component in an ontology learning framework. Using this component we have created a working ontology learning mechanism which, based on our experiment, performs fairly well.

6.2 Future Research

In this section we describe some of the work that can be done in order to improve the rule-engine.

6.2.1 Entire Documents Processing

During this master thesis we have focused on processing titles instead of entire news items. The reason for this is that titles are usually well formulated and to the point. Moreover titles can be processed in a limited amount of time. The drawback of this approach is that titles do not always contain all the needed information, which can be solved by processing the entire news item. However, this might introduce issues with respect to processing speed, which have to be solved first. Besides that, we do not know what the impact on the result exactly is, this means that additional measurements should be done.

6.2.2 Event Chains

In a lot of cases events are not isolated, but are part of a chain of events. An example is the process which results in one company buying another. In most cases this process starts with rumours which state that one or more companies are interested in buying a certain company. If these rumours turn out to be true, the companies might engage in a bidding war. After the bidding war the winning company acquires the company for sale. In most cases each phase in this process results in a flow of news items, which represent the successive events leading up to the final buy-event. It would be interesting to think about a way to formulate such chains of events in order to monitor the developments of specific domains over time.

6.2.3 Automatic Pattern Discovery

At this point patterns have to be manually formulated by a domain expert. It would be very useful if patterns could be learned (semi-)automatically. A possible approach might be to perform generalizations, based on co-occurrence analysis. By analyzing which class instances co-occur frequently with certain relations, patterns can be formulated. Domain expert can then validate these patterns and create actions which are associated with them. This results in a rule which can be used by our rule engine to mine news items.

Bibliography

- [1] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.
- [2] D. McGuinness and F. Hamelen. OWL Web Ontology Language. *W3C Recommendation*, April 2004. <http://www.w3.org/TR/owl-features/>.
- [3] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Journal of Web Semantics*, 4(1):14–28, January 2006.
- [4] A. Maedche and S. Staab. Ontology Learning. *International Handbooks on Information Systems*, pages 173–190, 2004.
- [5] W. Liu, A. Scharl, and E. Chang. Semi-Automatic Ontology Extension Using Spreading Activation. *Journal of Universal Knowledge Management*, 0(1):50–58, 2005.
- [6] C. Fellbaum. WordNet an Electronic Lexical Database. *Computational Linguistics*, 25(2):292–296, 1998.
- [7] G. Miller, C. Leacock, R. Teng, and T. Bunker. A Semantic Concordance. *Proceedings of ARPA Workshop on Human Language Technology*, 1993.
- [8] R. Navigli, P. Velardi, A. Cucchiarelli, and F. Neri. Extending and Enriching WordNet with OntoLearn. *Proceedings of Global Wordnet Conference*, pages 279–284, 2004.

- [9] M. Hearst. Automated Discovery of WordNet Relations. *WordNet: An Electronic Lexical Database and Some of its Applications*, pages 132–152, 1998.
- [10] Jun ichi Nakamura and Makoto Nagao. Extraction of semantic information from an ordinary English dictionary and its evaluation. In *COLING*, pages 459–464, 1988.
- [11] G. Geleijnse and J. Korst. Learning effective surface text patterns for information extraction. *Proceedings of the EACL 2006 workshop on Adaptive Text Extraction and Mining (ATEM 2006)*, pages 1–8, April 2006.
- [12] P. Cimiano and S. Staab. Learning Concept Hierarchies From Text With a Guided Hierarchical Clustering Algorithm. In *Learning and Extending Lexical Ontologies by using Machine Learning Methods*, pages 6–16, 2005.
- [13] Ian Horrocks et al. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission*, May 2004. <http://www.w3.org/Submission/SWRL/>.
- [14] Jena - A Semantic Web Framework for Java. <http://jena.sourceforge.net/>.
- [15] Sheffield NLP. Gate - general architecture for text engineering. Available online at: <http://www.gate.ac.uk/>,.
- [16] OpenNLP. <http://opennlp.sourceforge.net/>.
- [17] Hugo Liu. MontyLingua: An end-to-end natural language processor with common sense. Available online at: <http://web.media.mit.edu/~hugo/montylingua/>, 2004.
- [18] B. Aleman-Meza et al. Semantic Analytics on Social Networks: Experiences in Addressing the Problem of Conflict of Interest Detection. *Proceedings of the 15th international conference on World Wide Web*, 2006.

- [19] Philipp Cimiano and Steffen Staab. Learning by googling. *SIGKDD Explorations*, 6(2):24–33, 2004.
- [20] Joseph Weizenbaum. ELIZA – A Computer Program for the Study of Natural Language Between Man And Machine. *Communications of the ACM*, 9(1):36–45, January 1966.
- [21] ARQ A SPARQL Processor for Jena. <http://jena.sourceforge.net/ARQ/>.
- [22] Dan Klein and Christopher D. Manning. Fast Exact Inference with a Factored Model for Natural Language Parsing. *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, December 2002.
- [23] Dan Klein and Christopher D. Manning. Accurate Unlexicalized Parsing. *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 2003.
- [24] H. Cunningham. Gate, a General Architecture for Text Engineering. *Journal Computers and the Humanities*, 36(2):223–254, May 2002.
- [25] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS-00-10, Department of Computer Science, University of Sheffield, Nov 2000.
- [26] Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Computer and Information Science, University of Pennsylvania, 1998.
- [27] Maurice H. T. Ling. An anthological review of research utilizing montylingua, a python-based end-to-end text processor. *The Python Papers*, 1(1):5–13, 2006.
- [28] Diana Maynar, Milena Yankova, Alexandros Kourakis, and Antonis Kokosis. Ontology-Based Information Extraction For Market Monitoring And

Technology Watch. *Proceedings of The ESWC Workshop “End User Apects of the Semantic Web”*, Heraklion, Crete, May 2005.

- [29] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [30] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [31] K. Lee and T. Meyer. A Classification of Ontology Modification. *Proceedings of the 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia*, pages 248–258, 2004.
- [32] P. Buitelaar, P. Cimiano, and B. Magnini. *Ontology learning from text: Methods, evaluation and applications*, volume 123 of *Frontiers in artificial intelligence and applications*, chapter Ontology learning from text: An overview, pages 33–12. IOS Press, 2005.
- [33] M. A. Hearst. Automatic Acquisition of Hyponyms From Large Text Corpora. In *Proceedings of COLING-92, Nantes, France*, pages 539–545, 1992.
- [34] E. Alfonseca and S. Manandhar. Improving an ontology refinement method with hyponymy patterns. In *Language Resources and Evaluation (LREC-2002)*, pages 235–239, 2002.
- [35] A. Maedche and S. Staab. Ontology learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2).
- [36] E. Alfonseca and S. Manandhar. An unsupervised method for ontology refinement. *Proceedings of the First International Conference on General WordNet, Mysore, India*, January 2002.
- [37] M. Rajman and A. Bonnet. Corpora-based linguistics: new tools for natural language processing. *First Annual Conference of the Association for Global Strategic Information, Germany, Bad Kreuznach*, 1992.

- [38] Z. Harris. *Mathematical Structures of Language*. New York: Interscience Publishers John Wiley & Sons, 1968.
- [39] Marek Mahn and Chris Biemann. Tuning Co-occurrences of Higher Orders for Generating Ontology Extension Candidates. *Workshop on Learning and Extending Lexical Ontologies by using Machine Learning Methods in ICML*, 2005.
- [40] Chung Hee Hwang. Incompletely and Imprecisely Speaking: Using Dynamic Ontologies for Representing and Retrieving Information. In Enrico Franconi and Michael Kifer, editors, *KRDB*, volume 21 of *CEUR Workshop Proceedings*, pages 14–20. CEUR-WS.org, 1999.
- [41] A. Faatz and R. Steinmetz. Ontology enrichment with texts from the WWW. *Semantic Web Mining Second Workshop at ECML/PKDD-2002, Helsinki Finland*, August 2002.
- [42] Gregor Heinrich, Jorg Kindermann, Codrina Lauth, Gerhard Paaß, and Javier Sanchez Monzon. Investigating Word Correlation at Different Scopes – a Latent-Concept Approach. *Workshop on Learning and Extending Lexical Ontologies by using Machine Learning Methods in ICML*, 2005.
- [43] Hans Friederich Witschel. Using Decision Trees and Text Mining Techniques for Extending Taxonomies. *Workshop on Learning and Extending Lexical Ontologies by using Machine Learning Methods in ICML*, 2005.
- [44] S. Dumais, G. Furnas, T. Landauer, S. Deerweste, and R. Harshman. Using latent semantic analysis to improve access to textual information. *Proceedings, CHI '88*, pages 281–285.
- [45] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic analysis. *J. American Soc. Info. Sci*, 41(6):391–407, 1990.

- [46] A. Schutz and P. Buitelaar. RelExt: A Tool for Relation Extraction in Ontology Extension. *Proceedings of the 4th International Semantic Web Conference, Galway, Ireland*, November 2005.
- [47] Pavel Makagonov, Alejandro Ruiz Figueroa, Konstantin Sboyshakov, and Alexander Gelbukh. Learning a Domain Ontology from Hierarchically Structured Texts. *Workshop on Learning and Extending Lexical Ontologies by using Machine Learning Methods in ICML*, 2005.
- [48] R. Navigli, P. Velardi, A. Cucchiarelli, and F. Neri. Quantitative and Qualitative Evaluation of the OntoLearn Ontology Learning System. *Proceedings 20th COLING 2004, Geneva*, 2004.
- [49] Akshay Java, Tim Finin, and Sergei Nirenburg. SemNews: A Semantic News Framework. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 1939–1940. American Association of Artificial Intelligence, 2006.
- [50] Sergei Nirenburg and Victor Raskin. Ontological semantics, formal ontology, and ambiguity. In *FOIS*, pages 151–161, 2001.
- [51] N. Aussenac-Gilles. Supervised Text Analyses for Ontology and Terminology Engineering. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, February 2005.
- [52] A. Gomez-Perez and D. Manzano-Macho. OntoWeb Deliverable 1.5: A survey of ontology learning methods and techniques, May 2003. ONTOWEB Consortium.
- [53] The Rule Markup Initiative. <http://www.ruleml.org/>.
- [54] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *International Semantic Web Working Symposium (SWWS)*, 2001.

- [55] H. Eriksson. Using JessTab to Integrate Protégé and Jess. *IEEE Intelligent Systems*, 18(2):43–50, 2003.
- [56] V. Haarslev and R. Moeller. Racer: A core inference engine for the Semantic Web. In *2nd International Workshop on Evaluation of Ontology-based Tools (EON-2003)*, Sanibel Island, FL, 2003.
- [57] Evren Sirin and Bijan Parsia. Pellet: An OWL DL reasoner. In Volker Haarslev and Ralf Möller, editors, *Description Logics*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
- [58] C. Golbreich and A. Imai. Combining SWRL rules and OWL ontologies with Protégé OWL Plugin, Jess, and Racer. In *proceedings of the 7th International Protg Conference, Bethesda, MD*.
- [59] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. *W3C Candidate Recommendation*, April 2006. <http://www.w3.org/TR/rdf-sparql-query/>.
- [60] Andy Seaborne and Geetha Manjunath. SPARQL/Update - A language for updating RDF graphs. March 2007. <http://jena.hp1.hp.com/~afs/SPARQL-Update.html>.
- [61] Jethro Borsje and Hanno Embregts. SPARQLViz, 2006. <http://sparqlviz.sourceforge.net/>.
- [62] Christian Wartena. Apolda, 2007. <http://apolda.sourceforge.net/>.
- [63] M. Hepple. Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong, October 2000.
- [64] Eric Brill. A Simple Rule-Based Part of Speech Tagger. In *ANLP*, pages 152–155, 1992.