Ontological Representation of Texts, and its Applications in Text Analysis

by

Bent André Solheim *Kristian Vågsnes*

Master Thesis in
Information and Communication Technology

Agder University College

Grimstad, 26th May 2003

Abstract

For the management of a company, the need to know what people think of their products or services is becoming increasingly important in an increasingly competitive market. As the Internet can nearly be described as a digital mirror of events in the ”real“ world, being able to make sense of the semi structured nature of natural language texts published in this ubiquitous medium has received growing interest.

The approach proposed in the thesis combines natural language processing and ontologies to represent elements in texts and relations between them. These relations are used to determine what terms describe what objects and what spot the descriptions put the objects in. *Are the objects spoken of in a positive or negative manner?*

Throughout the report, it will be emphasized that ontologies can be applied to increase the ability of an automated agent to reason about texts. The more meta data that exist about a word or expression in a given context, the more deductions the agent will be able to make. One of the ontologies we developed allows the identification of statements that are considered either positive or negative relative to a given term.

Keywords: *Natural Language Processing, Ontologies, OWL, XML, Context Free Grammars, Positive/Negative Statements*

Preface

The project group consists of two master thesis students from Agder University College, Grimstad, Norway; Bent André Solheim and Kristian Vågsnes. It was supervised by Vladimir Oleshchuk, Agder University College, and Asle Pedersen, Intermedium.

Bent Andre Solheim is a Høgskole Ingeniør graduated from Agder University College. Presently, he is working part time at Intermedium as a systems designer, and doing his master thesis at Agder University College.

Kristian Vågsnes is also a graduated Høgskole Ingeniør from Agder University College. He is presently working as a senior developer in TimeBroker, and is finishing his master thesis at Agder University College.

Special thanks to Intermedium and our supervisors for letting us formulate our thesis definition freely, and for allowing us to work independently throughout the project period.

Grimstad, May 2003

Bent André Solheim and Kristian Vågsnes

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

1.1 Project Description

There are really several aspects about the spectacular eighth wonder of the world, the world wide web, that makes it virtually impossible for mere humans to grasp. The overwhelming amounts of data alone lay dead any audacious attempt to locate those golden nuggets of life altering information. To the rescue, industries, who's soul purpose are to locate and organize data for the users, have been emerging from the blurry and unfriendly ocean of information for years, taking names such as search engines, web crawlers and web portals. However, the limitations of the traditional repertoire of these services become apparent when users want answers to questions like *"Where are hotels with good skiing conditions located in Austria?"*, and then want the results ordered by how well the guests enjoyed staying at the hotel. Until recently, huge indexes and semi-intelligent algorithms based on a words location in the text, have been the most common ways to filter and grade pieces of information.

This report describes how *ontologies* can be applied to increase an automated agents ability to reason about natural language texts. It will become apparent that it is in fact possible to organize hotel surveys by how well the guests enjoyed staying there.

As a branch of increasing activity in the AI[1] community, the main application area of ontologies is knowledge representation. They moved out of the research labs in the '90s, and today many large international projects and enterprises, such as DARPA, W3C and CYCORP, are involved in developing, maintaining and representing ontologies.

Combining ontologies and natural language processing, it is possible to model the nature of positive and negative statements about an object, and then locate these or similar statements in a text. This way, we can identify the "temperature" of which a given object is spoken. Applying this technique in, for instance, a web crawler, key

[1] Artificial Intelligence

personnel could be notified when their business or products are described in a negative fashion on some web page.

1.2 Historical Summary

Ever since the first computers were made, the interest of computer-based linguistic technologies has grown more and more. This technology includes speech recognition and understanding, text retrieval and understanding, and the use of these methodologies in computer assisted language acquisition. However, human language is complex and information-rich. Therefore the work of creating automated computer systems is a long and expensive process; large amount of varied linguistic data, texts, lexicons and grammars are needed to make a robust and effective system.

Natural language processing is an extension of a grater unit called Artificial Intelligence (AI). AI has a long history, way back to 1000 B.C, starting with *I Ching*, one of the fundamental books of Confucianism. Since then, many organizations, projects and initiatives have been working with this ”problem“, but the expansion has grown most from the 50s and until now:

- **1946: ENIAC** - The world’s first electronic electronic digital computer.
- **1958: LISP** - John McCarthy created LISP. LISP is a object oriented programming language, made for simplifying the handling of arbitrary lists of arbitrary data objects.
- **Early 60s** - Beginning of semantic information processing; keyword $\rightarrow$ action, translation from English to formal language (a student solves math problems stated in English), database language queries.
- **1966: ELIZA** - ELIZA is a computer therapist. It uses keywords to generate possible responses. Functions as long as it is understood that it is some kind of stupid.
- **Early 70s** - Augmented Transition Networks (ATN); move towards representing the meaning of words (verb, noun etc)
- **1970: PROLOG** - Declarative languages like Prolog try to free programmers from thinking about what specific sequence of actions the computer will go through, allowing them instead to spell out a series of ’declarative’ statements that the computer is then able to determine the sequence from.
- **Mid 70s** - Moving towards Conceptual Information Processing; changing to top-down approaches (oppose to bottom-up).
- **80s** - Focusing on learning, integrated processing, subjective understanding. Text generation is improving.

- **1988: 386 processor** - The 386 chip brings PC speeds into a new level.
- **90s** - Statistical natural language processing.

The aim of the technology of linguistics might seem to be to replace the keyboards and mice as means of interaction with computer systems, with speech and handwritten input. However, it is likely that people will continue to write and talk; the intelligent systems are just becoming more and more important to us to ease our work day. This includes the gathering, organization, categorization, generation and presentation of data.

The latest years, *W3C* has been one of the leading institutions in creating standards on the web. With *Semantic Web* they will make standards that enable us to express ourselves in terms that our computers can interpret and exchange. *Linguistic Data Consortium* (LDC) is an open consortium of universities, companies and government research laboratories. It creates and collects distributed text, databases, lexicons and other resources for research and developments purposes.

There are also several other organizations, institutions and companies involved in this process, acting together or on their own.

1.3 Thesis Definition

Following is the thesis definition; the text on which all work is based;

"As the need for information is increasing, the problem of sorting out what is relevant, and what is of less interest, is of equal importance. The huge amounts of data makes it a time consuming job to discover, for instance, articles of real value. Therefore, humans are relying more and more on computers to sort and categorize data.

Working as a market analyst, you have to be aware of how the products of your company are spoken of by your customers and customers to-be. You have read product surveys from several sources, and the count can be overwhelming. The need to automatically sort these surveys by how your products are spoken of, can be a real time saver.

In this project, we will develop a method based on ontologies and computational linguistics that can tell if a text is positive or negative relative to a given term. A prototype that demonstrates how it works will also be developed. Rounding up, we will try to determine the accuracy of our algorithm, and identify the contexts in which such an approach is applicable, and in which it is not. It is intended that the system should be implemented and used in an information system."

1.4 Our Work

To our knowledge, there has been no official effort to produce taxonomies of expressions regarding what manner they describe an object, with the purpose of using them to automatically determine if a text is either positive or negative relative to a given term. There exist, however, a great deal of efforts, both open source and with limited access, to produce dictionaries, ontologies and linguistic tools with the purpose of making more sense of natural language texts.

We will suggest how to build ontologies from texts and how these can be combined with ontologies containing descriptions and their degree of positiveness and negativity. This combination allows deductions of how objects are spoken of in a text.

Throughout the project we will emphasize the importance of openness and will to cooperate to achieve results when working with natural language texts. As will become apparent, much of the work done in this area of research, and yet to be done, is extremely time consuming, and hence, requires funding beyond that allocatable by small and medium sized businesses. Our contribution to this ideal will be focusing our work on the English language, the currently most common one on the Internet. Also, we will require that the tools we use are free, preferably GPL[2] open source software, or similar.

1.5 Report Outline

The next two chapters (chapters 2 and 3) of this report are intended to give an introduction to the key concepts used when presenting the proposed system in chapter 4. Emphasis will be put on explaining what an ontology is, and its current applications. Also, an overview of key concepts and methods regarding natural language processing will be presented.

The proposed method is described in chapter 4. Following an explanation of how ontologies can be applied to represent relations between words in natural language texts, and to represent the general concepts of positive and negative terms, comes a suggestion of how to combine these two to extract the ”temperature“ of which an object is spoken. The implementation specifics are discussed in chapter 6.

Limitations with both the method and the prototype are presented in chapter 5. It will become apparent that the presented approach will benefit from interconnection with existing ontologies to be able to automate the identification of interesting terms in specific fields of endeavor. These chapters will also put the finger on performance problems with the prototype and the lack of *real* intelligence to identify such things as ironical or sarcastic formulations.

[2]GNU General Public License

Rounding up, chapters 7 and 8 ties the strings together and presents our conclusions and ideas for further work.

The appendices contain a presentation of different types of Part of Speech taggers and a comparison test of these, the tag set used in this report, and the implementation of the ontologies developed.

Chapter 2

Natural Language Processing

2.1 What is Natural Language Processing?

Ever since man started talking (and writing), the understanding of the language has been of great importance. Every word has its own meaning, but some words are also put together with other words to create new meanings. This knowledge, the knowledge about words, is called morphology, and is described in more detail in section 2.2.

Scientists have for centuries studied languages and their structures to make rules about how they should be used. However, even when all these rules are followed, it may be hard for us to understand the meaning of the written content.

In recent years, teaching computers to understand languages has become more and more interesting. This includes, for example, to enable an automated agent to understand what is written in a report, a web site or a book.

In the following sections, common concepts used in computational natural language processing will be introduced. This includes identifying sentence boundaries, parts of speech tagging, and grammatical analysis.

2.2 Morphology

Words are often built from smaller meaning bearing units, morphemes. The study about ways of making words is called morphology. Almost all words are put together by a stem, and an affix. The stem is the main morpheme of the word, having the actual meaning of the word, while the affix add additional meanings of various kinds.

There are four different kinds of affixes; prefix, suffix, infix and circumfix. Prefixes appear before the stem of a word, e.g. *un*do, suffixes appear after the stem, e.g. do*ing*, infixes appear in the middle of the word (not so often in English), and circumfixes

appear both at the beginning and in the end of the word, e.g. *un*do*ing*.

2.3 Sentence Boundary Disambiguation

Locating the end of a sentence is rendered difficult by the ambiguity of the usual sentence marks such as period('.'), comma(','), question mark('?') and exclamation mark ('!'). They do not always work as sentence breaks. For example, a '.' may be used in abbreviations, email-addresses, decimal points, and to indicate ellipsis. The question mark and exclamation point are more constant in their meaning, but they appear in some names, and may be used several times to emphasize a sentence boundary.

Many NLP tools require the text to be divided into separate sentences before they are able to do any sensible work with it. It is absolutely necessary for any human being or a computer to identify the different sentences in a context, because when a sentence boundary has been wrongly detected, its meaning will be improper.

Several techniques for detecting sentence boundaries are available. Following is a presentation of the most common ones.

2.3.1 The Direct Model

The Direct Model, uses hard coded routines the detect sentence boundaries. The algorithm is based on regular expression techniques. To increase the accuracy, it uses an abbreviation lexicon. Example 2.3.1 shows an example of a rule that describes a web address.

Example 2.3.1. *'[\-_A-za-z0-9.]+\.{dom}' WEBADDRESS*

2.3.2 Rule-Based Disambiguation

Hard coding is a non-dynamical developing model. In order to improve the Direct Model, the Rule-Based Disambiguation model has separated the description of sentence boundaries and the processing needed for their recognition. Regular expressions are used to represent the properties of sentences.

Example 2.3.2. *Sentence* $\rightarrow$ *All Word PERIOD PUNCT PERIOD ;*

describes a sentence consisting of a number of words, followed by a period, and unspecified punctuation character and another period [1]. The rule is able to capture ellipsis and other combinations like .". (2.3.3)

Example 2.3.3. *The man said "I would like that.".*

2.3.3 Maximum Entropy

The Maximum Entropy algorithm is a statistical way of finding sentence boundaries. It functions like this:

- The different potential sentence boundary tokens[1] is found, by scanning the text for sequences of characters separated by white spaces (tokens) containing either exclamation mark ('!'), period ('.') or question mark ('?').
- A joint probability distribution *p* is estimated to the token and its surrounding context. The distribution is made by the formula in equation 2.1.

Only specific evidence of sentence boundaries are considered by this model. This evidence may indicate that a sentence boundary is determined, and is based on prior linguistic knowledge about contextual features of text. Features unknown to the model, will be assigned a uniform distribution. The model will therefore be maximally uncertain about features of text for which it has no prior knowledge [1].

The formula of the distribution is given by

$$p(b,c) = \pi \prod_{j=1}^{k} \alpha_j^{f_j(b,c)}, b \in \{no, yes\} \tag{2.1}$$

where the α_j's are the unknown parameters of the model, and where each α_j is the weight of each features [2]. The probability of seeing an actual sentence boundary in the context c, is thus given by $p(yes, c)$.

The unknown parameters (weight) are discovered during training. Calculation of these features is computationally expensive, because each feature must be evaluated several times. For each iteration the task must be performed, and hundreds of iterations may be required to until the weights have converged satisfactory.

2.4 Text Tagging

Part of speech tagging is used for the preparation of texts before subjecting them to syntactic analysis, or in other words, the problem is to find the right tag for the right word, since one word may have several meanings.

For example, consider the word "race"; E.q *It was a hard race* and *I race with my brother every Wednesday*. In the first sentence, the word race is a noun, meaning a long run, while in the second sentence the word race is a verb. Several methods have

[1] A token in the training data is considered an abbreviation if it is preceded and followed be whitespace, and it contains a . that is not a sentence boundary.

been developed to disambiguate words and from the study of these methods, some common points have emerged:

- A word may only have limited set of tags, which can be found in a dictionary or through a morphological analysis of the word.
- When a word has more than one possible tag, rules are used to find the correct tag from the local context.

There exists a number of different software taggers. Some are described in Appendix A. Depending on the type of tagger, some of them require training, i.e. a feed of manually correctly tagged texts. This is done through an implementation specific training method for each tagger.

Part-of-speech (POS) tagged training-text need to be accurate. If the taggers are trained in a wrong way, wrong tags will be assigned wrong words when tagging untagged text. Large amounts of manually tagged text are used as training material to make the taggers as accurate as possible [3].

For a specific sentence there is only one correct way to tag the different words. Consider a set of words W=w_0+w_1+.... + w_n, and a set of tags T=t_0+t_1+....+t_n of the same length. For the sentence there is a correct alignment between these two sets (W, T). The tagging procedure ϕ selects the right sequence of tags and selects the correct alignments for each sentence:

$$\phi : W \rightarrow T = \phi(W) \tag{2.2}$$

The quality of a tagging procedure has two measures:

- At sentence level; percentage of sentences correctly tagged.
- At word level; percentage of words correctly tagged.

The percentage at sentence level is generally lower than at word level. This is because all words must be tagged correctly for the sentence to be tagged correctly [4].

There exists a few different methods for tagging text. Some of them are based on the same ideas of text-processing as the methods for sentence boundary detection in section 2.3. In the following sections some of these are described in detail.

2.4.1 Rule-Based

Rule-based algorithms were the earliest form of automatically assigning part-of-speech. It was divided into two stages; first using a dictionary to assign each word a list of potential POS (figure 2.1), second a list of hand-written rules for disambiguate the first list into a list of one tag for each word [5].

WORD	POS	ADDITIONAL POS FEATURES
look	V	Present
look	N	singular
race	N	singular
race	V	present
racist	ADJ	
racist	N	singular
racially	ADV	

Figure 2.1: Example of a dictionary of possible words and tags

2.4.2 Probabilistic

First time one used probabilities in part of speech tagging, was in 1965, so the method is quite old. The basics behind a probabilistic tagger is to ”pick the most likely tag for this word“. More detailed descriptions of different probabilistic algorithms are described in the following sections.

Maximum Entropy

In many ways, the maximum entropy probabilistic POS tagging algorithm functions the same way as the maximum entropy algorithm described in section 2.3.3, but the model is of course tailored for, in this instance, the text tagger purpose.

The tagger contains a set ($\mathcal{H}$) of possible word and tag contexts (*histories*), and a set ($\mathcal{T}$) of allowable tags. The probability-model is defined over $\mathcal{H}x\mathcal{T}$.

$$p(h,t) = \pi\mu\prod_{j=1}^{k}\alpha_j^{f_j(h,t)} \tag{2.3}$$

where π is a normalization constant, $\mu, \alpha_1, \ldots, \alpha_k$ are the positive model parameters and $\{f_1, \ldots, f_k\}$ are known as ”features“ $f_j(h,t)\ \epsilon\ \{0,1\}$ [3].

Hidden Markov Model

The Hidden Markov Model (HMM) chooses tags that maximizes the following formula:

$$P(word|tag) * P(tag|previous\ n\ tags) \tag{2.4}$$

A HMM-tagger tags whole sentences rather than single words. It uses a viterbi approximation [6], and chooses the most probable tag sequence for each sentence. The algorithm tries to find the sequence of tags T = $t_1, t_2, t_3, \ldots, t_n$ to the given sequence of word (W):

$$\hat{T} = \underset{T \in \tau}{argmax} P(T|W) \text{ [5]} \tag{2.5}$$

The HMM is trained by hand-tagged data, but it is also possible to use unlabeled data to train this model using the EM algorithm[2]. These taggers have a dictionary which holds lists of which tags can be assigned what words.

2.4.3 Transformation Based

Transformation Based Tagging draws inspiration from both the rule-based and probabilistic taggers. It is an instance of the Transformation Based Learning (TBL) approach to machine learning. TBL has, like rule-based taggers, a list of which tags may be assigned to which words. But it is also a way of training a program, where rules are automatically created from the data (see figure A.2). The model is trained by a pre-tagged training corpus.

The TBL can be described as follows; when it tries to tag a word[3], it first collects all the tags assigned to this word. Then it starts with the broadest rule, which will apply to most of the tags. Next it tries a more specific rule, and then again an even more specific rule until TBL can consider which tag to choose.

2.5 Text Analysis

When people read an article, they have to recognize what is written to understand the meaning of it. The brain has through years of reading developed a way of understanding text. The more one reads, the easier it is to recognize patterns in a text. Reading a sentence that is ”out of order“, may not make any meaning.

To determine the meaning of a sentence is a complicated task. Many rules are made about how a sentence may be build, to get proper content. For example, the sentence ”I on would Tuesday like to cinema go“ is hard, or may even be impossible to understand, since the order of the words is incorrect.

[2] [5] chapter 7 and Appendix D

[3] word in a sentence. This tagger tags sentences as a whole.

2.5.1 Non-Terminals

The rules about how to create proper sentences are made of a set of variables. These are called non-terminals. Basically the non-terminals consist of all the different tags the chosen text tagger uses. This could be VB (verb), PRP (personal pronoun), etc.[4]

Along with all these tags, a non-terminal also contains something called phrases. A sentence is made of a noun phrase and a verb phrase. Then again, these phrases are build from different kinds of smaller objects. In table 2.1 all the phrases used are listed.

Table 2.1: Phrases

PHRASE NAME	DESCRIPTION
Noun Phrase ***NP***	A sequence of word surrounding at least one noun (e.g three parties from Brooklyn)
Verb Phrase ***VP***	Consists of a verb followed by assorted other strings. (e.g prefer a morning flight)
Prepositional Phrase ***PP***	Preposition followed by a noun phrase. (e.g from Los Angeles)
Nominals ***NOM***	A nominal is an order of one or more nouns.
Adjective phrase ***AP***	Consists of one ore more adjectives.

2.5.2 Context Free Grammars

When a sentence is to be analyzed, the sentence has to be split into many smaller parts, like the noun phrase and verb phrase mentioned in section 2.5.1. The sentence has root, meaning the beginning of the sentence text, and then there have to be some elements to make the sentence correct grammatically. From the sentence root, there is made a hierarchy (or a tree). Context free grammars, or CFG is a powerful tool that is perfect with respect to the operation of creating this hierarchies.

When computers are taught to understand sentences, they have to know about a number of rules or grammars. One way of creating certain grammars is to use context free grammars. A CFG consists of leaf values and node values. A node value might be saying that a Proper Noun is a Noun phrase (***NP***), and a leaf value could be saying that 'the' means a determiner (***DT***).

Symbols used in CFG are divided into two classes; symbols corresponding to words

[4] All these are listed in guidelines to the Treebank Project [7].

L_0 = "I am a student at Agder University College"

Figure 2.2: A given sentence L_0

in the language ('bicycle', 'work') which are called terminal symbols, and the on-terminals which are the set of rules that introduce these terminal symbols. The set of rules that say which words are combined with which non-terminals are called a lexicon (figure 2.3), whilst the rule for sentence building are called grammars, or productions (figure 2.4).

Personal pronoun ***PRP***	$\rightarrow$	I
Verb present ***VBP***	$\rightarrow$	am
Determiner ***DT***	$\rightarrow$	a
Noun singular ***NN***	$\rightarrow$	student
Preposition ***IN***	$\rightarrow$	at
Proper-Noun singular ***NNP***	$\rightarrow$	Agder, University, College

Figure 2.3: The Lexicon of L_0

A CFG like the one in figure 2.4 is used to describe a formal language. A sentence in a certain language contains a set of strings that are derivable from a designated *start symbol*, which each grammar must have. The start symbol is often called ***S*** (the sentence node). If a sentence cannot be derived from the given formal grammar, the sentence is not in the language defined by that grammar, and is referred to a ungrammatical. The use of formal languages to model natural language is called *generative grammar* [7], since the set of possible sentences is generated by the grammar.

To summarize, a grammar could be described like this:

$$G = (N, \Sigma, S, P)$$

where N is a set of non-terminal symbols, Σ is a set of terminal symbol (disjoint from N), P is a set of production (each on the form $A \rightarrow \alpha$, where A is a non-terminal and α is a string of symbols from the infinite set of strings $(\Sigma \cup N)$), and S is a designated start symbol.

2.5.3 Sentence-level Construction

Sentences are build in various ways. They can be declarative or imperative, or they can have structures such as questions. The *subject* of a sentence is located differently de-

S	$\rightarrow$	***NP VP***	I + am a student at Agder University College.
NP	$\rightarrow$	***NN***	I
	|	***DT NN***	a + student
NOM	$\rightarrow$	***NNP***	Agder | University | College
	|	***NOM NOM***	Agder + University + College
VP	$\rightarrow$	***VBP***	am
	|	***VP NP***	am + a student
	|	***VP PP***	am a student + at Agder University College
PP	$\rightarrow$	Preposition ***NP***	at + Agder University College

Figure 2.4: The grammar of L_0, and example of phrases of each rule.

pending on the sentence structure. It is important to know where to locate the subject, since the subject often is one of the most important words in a sentence.

Declarative

Sentences with declarative structure is made of the form ***S*** $\rightarrow$ ***NP VP***, where the subject is in the noun phrase.

Example 2.5.1. *I love Norwegian country.*

Imperative

The imperative statements are often used for commands and suggestions. The structure usually begins with a verb phrase and have no subject (***S*** $\rightarrow$ ***VP***)

Example 2.5.2. *Give me that book!*

Yes-No-Question

Questions that are possible to answer with a simple yes or no, are called yes-no-questions. They often (not always) have an auxiliary[5] verb before the subject noun phrase, followed by a verb phrase. (***S*** $\rightarrow$ ***aux NP VP***)

Example 2.5.3. *Do you know my name?*

[5]Auxiliary verbs are also called helping verbs, and includes modal verbs (*can, may*), the perfect (*shall*), the progressive (*be*) and the passive auxiliary (*be*).

Wh-Question

This is one of the most complex sentence structures. It is so-called because it includes a wh-phrase, that is a phrase that contains a wh-word (who, what, whom). These questions are grouped into two classes of sentence-levels: The *wh-subject-question* and *wh-non-subject-question*

The first one is almost the same as the declarative structure, except that the first noun phrase contains some wh-words (***S*** $\rightarrow$ ***Wh-NP VP***).

Example 2.5.4. *What is the time?*

In the second structure, the wh-noun phrase does not include the subject of the sentence. The main subject appears in another noun phrase (***S*** $\rightarrow$ ***wh-NP aux NP VP***).

Example 2.5.5. *Which bus do you prefer to town?*

2.5.4 Sentence Parsing

A sentence may be divided into many smaller contexts. This could be verbal, subject, object. The work of analyzing of sentences is called parsing. The parsing process of a sentence may result in a parse-tree. An example of a parse-tree is shown in figure 2.5.

So far, only the preprocessing that has to be done before the sentence-parsing can commence has been described. The text has been split into sentences, the sentences have been tagged, and rules are made about where the different tags appear in sentences and how the tags work together. The last problem is to iterate through the different sentences to analyze what is the subject, verbal, object and so on.

The way this is done, a parser searches through the space of all possible parse-trees to find the correct parse-tree for the sentence [7].

Figure 2.5 contains a parse-tree of the sentence *"I am a student at Agder university*". A sentence can generate many different parse-trees, and the problem is to choose the correct one. A parse tree has two kinds of constraints:

- The input-sentence itself. In figure 2.5 there will be exactly seven leaves (seven words in the sentence).
- The grammar. The parse-tree can only have one root, which must be the start symbol ***S***.

Together with these constraints we have two parsing strategies: *top-down* and *bottom-up*.

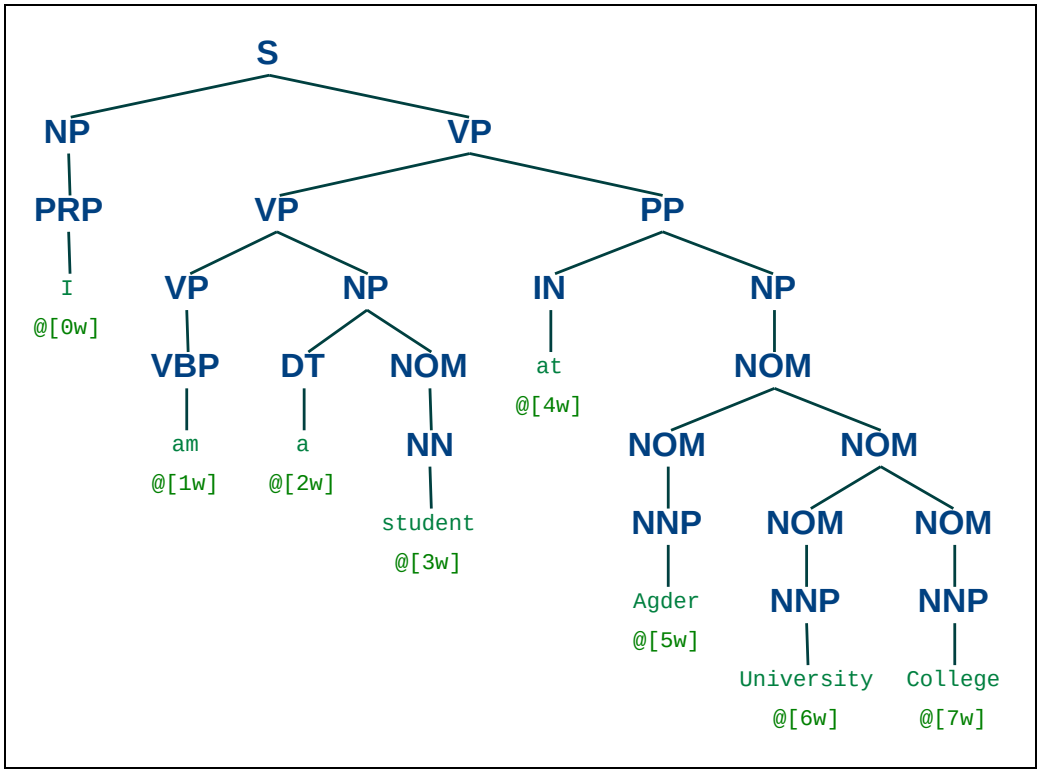


Figure 2.5: A parse-tree of sentence L_0

Top-Down

A top-down parser tries to find a parse-tree by building from root node (***S***) to the leaves. First it finds all the grammar that expands ***S***. Then it expands the constituents in these trees. The rest of the search is performed recursively, meaning that new sets of expectations are provided for each ply of the search space. It finishes when the bottom of the tree is reached. Trees which fail to match all the words in the input are rejected.

Bottom-Up

In bottom-up parsing, which is the earliest known parser-algorithm, the parser starts with the words from the input, and tries to build trees from the words up, by applying rules from the grammar one at a time [7]. A parse-tree is build if it manages to fit the rules to the words. The tree starts at ***S***, like in the top-down algorithm.

Chapter 3

Ontology

3.1 What Is an Ontology?

Ontology is a concept that has existed for centuries as a discipline of philosophy where it refers to the subject of existence. It has also been confused with epistemology; about knowledge and knowing. However, in computer science, the word has a very different meaning. Evolving from semantic network notions, modern ontologies are stated to be *a specification of a conceptualization* [1]. This means that an ontology accounts for the possible and allowed terms and relationships that can exist in a conceptualization. Also, ontologies constrain the intended meaning by stating axioms that limit the possible interpretations for the defined terms.

Ontologies are being built today to make a number of applications more capable of handling complex and disparate information. When the semantic distinctions humans take for granted are important to an application's intention or use, ontologies are very effective. This includes for instance handling expertise embedded in domain-specific knowledge repositories, or dealing with common sense in natural language texts [9].

The difference between a good and a bad ontology is how well it maps to the intended model. As figure 3.1 illustrates, mapping well to the intended model implies not only to let the ontology be able to capture all the intended meanings, but to exclude all the unintended ones; if a class is a description of a group of objects that share some properties, and the classes Man and Woman are stated to be disjoint, a reasoner can deduce that an object of the class Man is not also of the class Woman. This way, ontologies establish a joint terminology between members of a community of interest, being either human or automated agents.

Traditionally ontologies have been used in knowledge representation and library science, but in recent years they have been included in most expert systems, and used for both knowledge sharing and reuse. As of this writing, there is active research on the

[1] Tom R. Gruber [8]

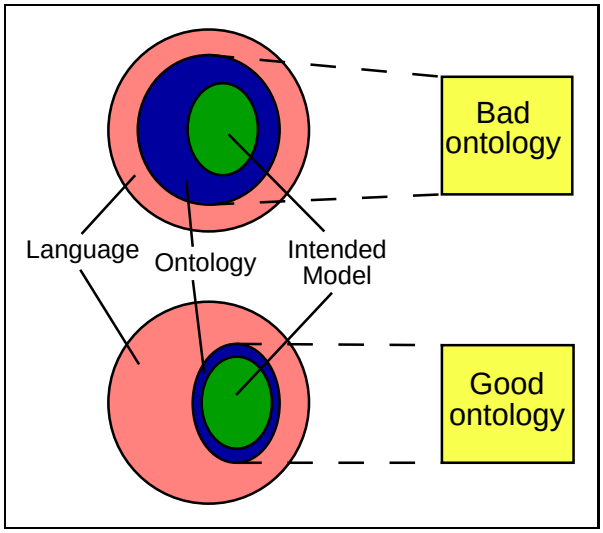


Figure 3.1: The difference between a good and a bad ontology.

role of ontologies in fields such as knowledge representation, knowledge engineering, database design, information retrieval, natural language processing, and the semantic web [10].

3.2 Building Ontologies

How one goes about building an ontology reflects what is to be described. The structure and implementation of simple lexicons will differ from those of organized taxonomies with hierarchically related terms and distinguishing properties. The ontologies will also vary in the purpose of their content; typically we have upper level ontologies describing the basic concepts and relationships in the natural language of any domain, and domain ontologies describing specific fields of endeavor, like telecom and DAC.

Generally, ontologies can be divided into two logical parts; the terminological component and the assertional component [11]. The terminological component is analogous to an XML schema and defines the terms, classes (concepts), and structure of the ontology's area of interest (sometimes this part is referred to simply as the ontology). The assertional component uses the terminological concepts to define instances and individuals. For example, as figure 3.2 shows, are Bob and Alice instances (or individuals) of the classes Man and Woman, respectively, in the People-ontology. Sometimes, these assertions are called a knowledge base (KB) and may include facts about individuals that are members of classes, and derived facts that are not literally present in the textual representation of the ontology, but logically implied by the semantics.

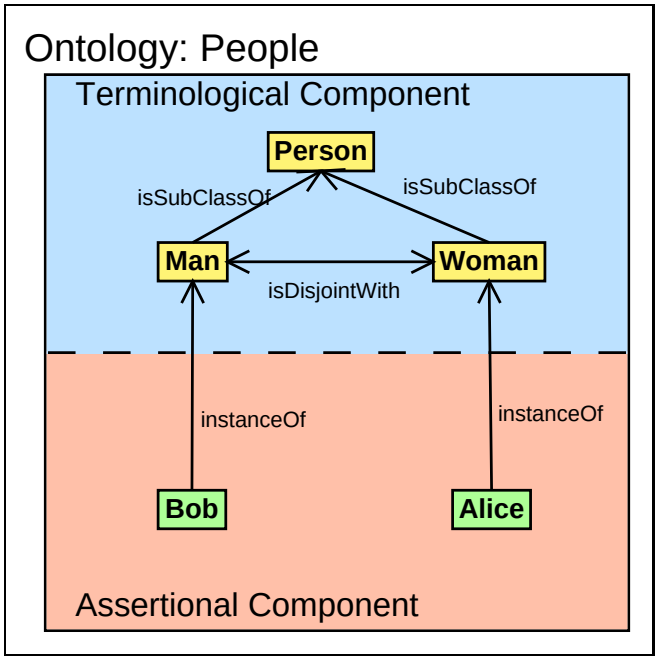


Figure 3.2: The figure shows the terminological and assertional components of an ontology as two different logical parts.

3.3 Representing Ontologies

To represent ontologies, a representation language is required. Several different languages are available, some based on standardized syntax, others not. For web applications, it is important to use standardized syntax to be able to exchange ontologies efficiently and to simplify the task of writing parsers. As XML [12] the past few years has emerged as the standard for exchanging data on the web, a number of ontology representation languages are based on this format. In general, ontology languages are increasingly relying on other W3C [13] technologies as well, like RDF Schema as language layer, XML Schema [14] for data typing, and RDF [12] to assert data [11].

Ontology languages are usually compared regarding their expressiveness, i.e. *what can be said* about a specific knowledge domain. According to Gruber [8], knowledge in ontologies can be specified using five kinds of components; concepts, relations, functions, axioms and instances;

- Concepts, or classes, are used in a broad sense. They can be anything that it is possible to say something about; a task, action, strategy, etc. Usually, the concepts in an ontology are organized in taxonomies.
- Relations represent a connection or interaction between concepts in the domain (or other domains).
- Functions are a special kind of relation where the last argument is unique for a

list of values of the n-1 preceding arguments.

- Axioms model statements that are always true. Their main purpose is to constrain information, i.e. ensure unambiguity, and hence verify information's correctness, but they can also be used to deduce new information.
- Instances or individuals represent the elements of concepts in the domain. Sometimes these are maintained separately as a knowledge base.

Following is an overview of some commonly used ontology representation languages. Only their main features will be presented, but references to their specifications or to articles describing them are provided.

3.3.1 Common Logic Standard

Common Logic Standard [15](CL) is a framework for a family of logic-based languages. Its semantics is a superset of that of many other logic-based systems. This way, semantics of different languages can be represented with CL, and hence information can be preserved when interchanged among hetrogenous systems. In fact, CL facilitates interoperability among formal specification languages like UML [16] and semantic web initiatives like OWL (see [17] and section 3.3.6).

The specification of KIF (Knowledge Interchange Format) is a part of the CL Project. It is being developed to solve the problem of hetrogenity of languages for knowledge representation. Languages like Ontolingua (see [18] and section 3.3.2) is based on this format.

3.3.2 Ontolingua

Ontolingua [18] is a language based on KIF and on the Frame Ontology (FO), and it is the ontology-building language used by the Ontolingua Server.

As KIF is an interchange format, it is tedious to use for specification of ontologies per se. The FO, built on top of KIF, is a knowledge representation ontology that allows an ontology to be specified following the paradigm of frames, providing terms such as class, instance, subclass-of, instance-of, etc. The FO does not allow to express axioms; therefore, Ontolingua allows to include KIF expressions inside of definitions based on the FO. Ontolingua allows to build ontologies in any of the following three manners:

- Using exclusively the FO vocabulary (axioms cannot be represented).
- Using KIF expressions.
- Using both languages simultaneously.

3.3.3 Loom

Loom [19] is a high-level programming language and a environment intended for use in constructing expert systems and other intelligent application programs. The main part is a knowledge representation system, which in turn is used by the declarative portion of the language for deductive support.

Loom allows utilization of logic programming, production rule, and object oriented programming paradigms in a single application because of the high degree of integration between the different main parts.

3.3.4 Resource Description Framework (RDF)

The Resource Description Framework is a foundation for processing meta data, intended to be used to provide meta data about data contained on the Web. This way, RDF can be used by search engines, in cataloging, in content rating, and similar.

One of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner. See [12] and [20] for more information on RDF.

3.3.5 DAML+OIL Ontology Markup

The DARPA Agent Markup Language [21] program officially started autumn 2000. Its goal was to facilitate the Semantic Web with providing a language that remedied the limitations of XML regarding relation representation. Building on RDF and RDF Schema, it provides a rich set of constructs with which to create ontologies and to markup information so that it is machine readable and understandable.

3.3.6 OWL - Web Ontology Language

The Web Ontology Language (OWL) [17] is a W3C standardization effort to create a semantic mark up language for publishing and sharing ontologies on the Web. It is derived from the DAML+OIL Ontology mark up language (see section 3.3.5), and is hence based on other W3C standards; XML, RDF and RDFS, but facilitates greater machine readability of web content than these technologies by providing an additional vocabulary for term descriptions.

OWL provides two subsets intended for users not requiring the functionality of the entire languages;

The OWL specification is, as of this writing, a work in progress.

- The *OWL Lite* subset was mainly designed for easy implementation and to provide users with a soft introduction to OWL. It only supports a subset of the OWL language constructs, but broadens the modeling capabilities of RDFS by adding some common features used when extending ontologies with vocabularies and thesauri. OWL Lite tries to capture many of the most common features of OWL and DAML+OIL.
- *OWL DL (Description Logic)* is primarily a language subset that has desirable computational properties for reasoning systems. It includes the entire OWL vocabulary, only with a few constraints. Most prominent among these are that class identifiers cannot be a properties or individuals at the same time, and that properties cannot be individuals.
- *OWL Full* is the name used to refer to OWL as a whole. All constructs of the language are available.

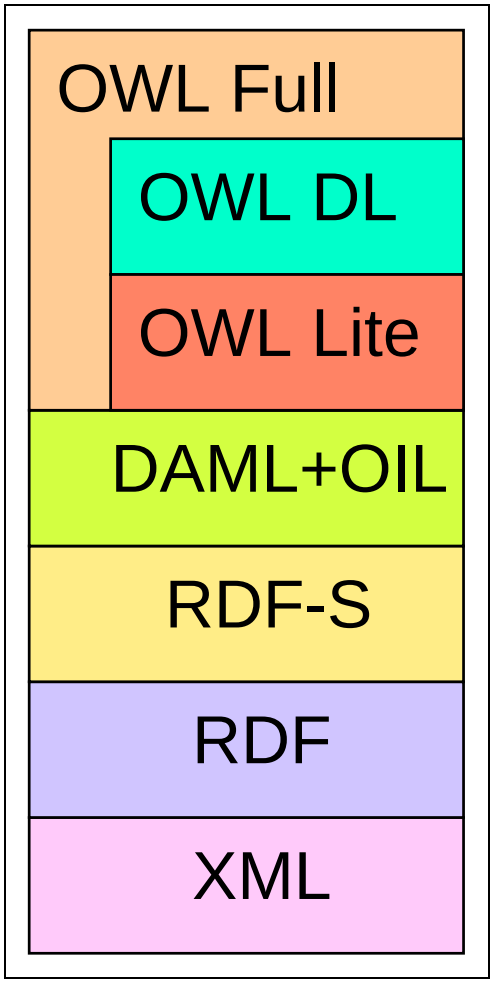


Figure 3.3: The different technologies making up OWL.

Chapter 4

The Proposed Solution

Having discussed the most important aspects of what ontologies and natural language processing are, this chapter will cover the proposed solution to the presented problem; how to use ontologies to identify how an object is spoken in a natural language text.

In the description of the solution, the terms warm and cold will be used when describing how an object is spoken. These are considered more general than, for instance, the terms positive and negative. Positive and negative descriptions will always be relative to the individual receiving them. An article reporting that the new flagship mobile phone from Ericsson contains a lot of software bugs, and is therefore considered a bad phone, is of course negative for Ericsson, but could be considered positive for their competitors. A warm or a cold description is not relative to the receiver. The new phones from Ericsson being described as bad, is a cold description, regardless of who is reading the review.

4.1 Using Ontologies

As mentioned in chapter 3, representing knowledge using ontologies has many advantages. Having a plain text corpus described using some ontology representation language would enable automated agents to reason about the contents of the text. This is possible, because the ontological representation is a representation of the knowledge in the text; *what does this text really mean*? Instances of words, the descriptive relations between them, and the logically implied information deducted from these relations, allow agents to answer questions such as “does this review speak of my product in a positive way?”, because the ontological representation of the text makes the terms that describe the product in question available. These terms can then be looked up in other ontologies to see whether or not they have a temperature that is either warm or cold. The more relations between the words, or meta-data about the them, the more questions are possible to answer, and the more precise the conclusions will be.

The sections in this chapter describe the proposed method of how to use ontologies to determine the way an object or term is spoken of in a text. To do this, three ontologies will be developed. One ontology containing descriptions; words and expressions, one containing all the terms used when determining the temperature of a description, and one containing concepts and relations required to describe relations between elements in a text.

4.2 Method Overview

As texts usually are unavailable represented as ontologies, they must be preprocessed to obtain such a representation. Consider the following text containing three sentences extracted from a review of the Ericsson P800 cellular phone:

> Reception proved acceptable, although signal strength was poor in areas of weaker network coverage$_1$. The phone was occasionally failing to register the SIM card$_2$. Voice quality was average$_3$.

Figure 4.1: Example Text

An ontological representation of this text would describe the different elements making it up and relations between them, identifying the meaning of the different words. The graph shows schematically how this representation could look using the second sentence from the text in figure 4.1 as an example:

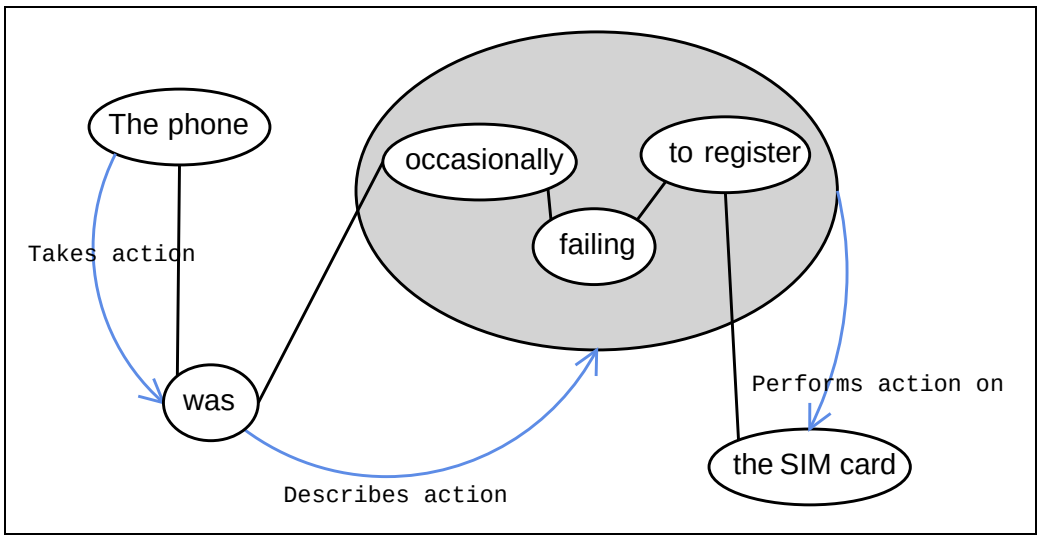


Figure 4.2: Example of how an ontological representation of the text could look.

It would also make sense to include information about which part of speech the different words are, and which particular phrase they belong to.

The ontology containing the concepts and relations used in the ontological representation of text will be called *the Text Ontology*. Texts represented as ontologies are then used by a reasoner to determine how objects are spoken of. The reasoner utilizes another ontology that defines temperatures of descriptions, called *the Description Temperature Ontology*.

By separating the ontological representation of the text and the description temperatures, the ontological representation can be used in other areas of application as well, without mixing with description temperatures.

Considering this, the method will consist of two processing units:

- **The PreProcessor** - The PreProcessor is the processing unit that will perform all the natural language processing; split the text into sentences, tag the sentences and establish relations between words, and generate the ontological representation of the text.
- **The Reasoner** - The reasoner is the processing unit that will use the ontological representation of the text and the ontological representation of object descriptions to determine how a particular term or object is spoken of. The results presented by the Reasoner will be called *deductions*.

and three ontologies;

- **The Text Ontology** - The ontology that contains the definitions of how relations between words and expressions in texts should be represented. The assertional component of this ontology is the preprocessed texts.
- **The Descriptions Ontology** - This is a simple lexicon containing words, expressions or phrases that are considered descriptions. Only terms found in this ontology will be identified as descriptions by a reasoner.
- **The Description Temperature Ontology** - Contains definitions of what makes a description. warm and what makes it cold. The assertional component of this ontology contains the temperature of the different words and expressions the Reasoner will be able to reason about.

The following figure (figure 4.3) describes how the five defined components are connected.

In the next sections, the components will be described in more detail. First the Text and Descriptions ontologies will be presented with all the terms, concepts and relations required to represent a text, followed by the functionality of PreProcessor. Then the Description Temperature Ontology will be defined. Finally, the approach to use the assertional components of all three of the defined ontologies to discover when an object of interest is described either in a warm or cold fashion will be outlined in the section about the Reasoner.

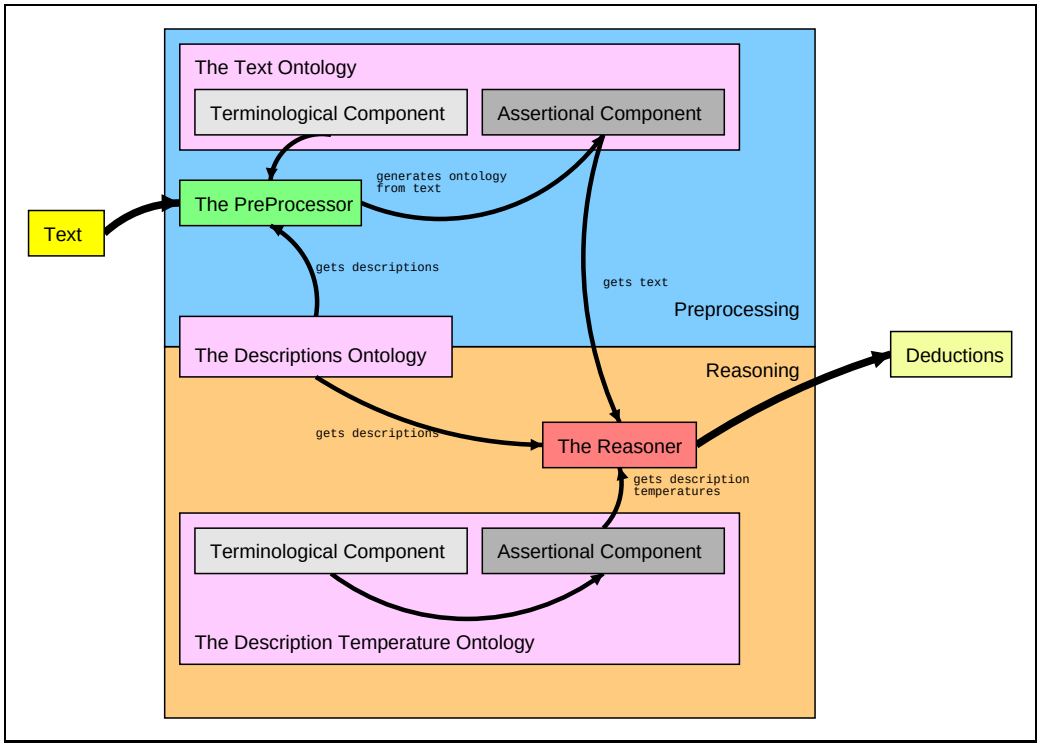


Figure 4.3: Overview of the defined components of the proposed method.

4.3 The Text Ontology

The next section will describe how a natural language text can be represented using ontologies. This includes definitions of elements making up a text and (some of) the relations between them.

As described in chapter 3, an ontology consists of two logical parts; the terminological component and the assertional component. In our approach, the assertional component of the Text Ontology will be made up of the analyzed texts and the relations between the words making them up. The terminological component defines how these texts should be represented.

4.3.1 The Terminological Component

To be able to represent relations between different elements in a text, the different types of elements must first be identified and formalized. Then definitions of meaningful relations between them can be made. Following is a list of concepts required in the ontology describing text[1].

[1]The definitions are based on the descriptions of the words found in WordNet[22].

- **Text** - The words of something written. In our context, we redefine the term text, narrowing it to be only the subset of texts consisting of one or more sentences.
- **Sentence** - A string of words satisfying the grammatical rules of a language. Per definition, one or more sentences make up a text.
- **Word** - Words are the blocks from which sentences are made.
- **Word class/Part Of Speech (POS)** - Category of words intended to reflect their functions in a grammatical context.
- **Phrase** - An expression forming a grammatical constituent of a sentence but not containing a finite verb.
- **Expression** - A group of words that form a constituent of a sentence and are considered as a single unit.

Note 4.1. The definition of the text concept is quite broad. It is not taken into consideration that a text sometimes consists of elements to separate it into logical parts, like headlines and paragraphs, nor is it distinguished between genres of texts, like articles, reports or books. It is assumed that a text can be considered a logical unit, and analyzed thereafter.

Note 4.2. POS is a concept comprising all the names of the categories of words. A word will always be of one of these categories in a given context. The category noun, for example, will be an instance of the part-of-speech concept. There exists no explicit relation between the word-concept and the POS-concept. Only between words and subclasses of the POS-concept.

Having defined the main concepts in the Text Ontology, the relations between them must be made out. The following observations work as a foundation for the definitions (the names of the relations will be written using mixed case, except for the initial letter, which is lower case);

- A text is *madeUpOf* sentences.
- A sentence is *madeUpOf* words.
- The *madeUpOf* relation is transitive; since a text is made up of sentences and a sentence is made up of words, a text is made up of words.
- An expression is *madeUpOf* words.
- A word can *describe* another.
- An expression can *describe* another.
- A word can *describe* an expression, and vice versa.

- A word *isOf* a word class.
- An expression can be a phrase. Therefore, sometimes, an expression *isA* phrase.
- A word can *referToTheSameAs* another word (The phone → it).
- An expression can *referToTheSameAs* a word (The new phones from Nokia → they).

Figure 4.4 illustrates the concepts and the relations between them.

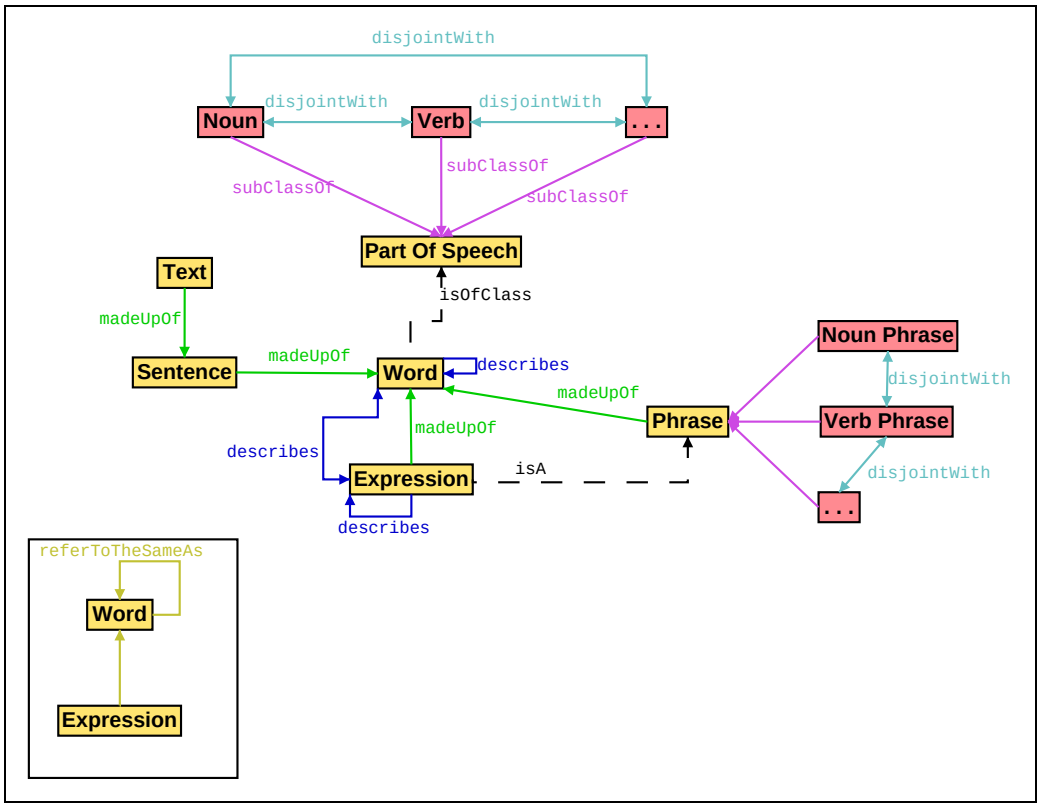


Figure 4.4: A schematic representation of the concepts and relations in the Text Ontology.

Note 4.3. All the Part Of Speech subclasses are disjoint; if a word is of one class, it can not be of any of the others.

The concepts (or terms) defined above will be the base of the implementation of the terminological component of the Text Ontology. The assertional component of the Text Ontology will be the ontological representation of texts.

When the desired natural language processing of a text has completed, the relations and information produced by the processing is represented as an ontology, based on the terminological component of the Text Ontology.

4.3.2 The Assertional Component

As mentioned above, the assertional component of the Text Ontology will be the ontological representations of texts. The PreProcessor (see section 4.5) uses the terminological component as a specification of how to represent the texts as ontologies. The assertional component will be under constant modification because texts will be added as they are processed. This allows a distributed development of the Text Ontology.

As an example, consider the third sentence from figure 4.1; ”Voice quality was average“. The part of speech tagging of this sentence yields;

WORD	PART OF SPEECH
Voice	Proper Noun, singular
quality	Noun, singular
was	Verb, past tense
average	Adjective

Table 4.1: The parts of speech for the words in the sentence ”Voice quality was average.“

”Voice quality“ is the noun phrase of this sentence, while ”was average“ is the verb phrase. The verb phrase is the action the noun is taking, i.e. the verb phrase tells something about the noun. ”was“ connects the adjective ”average“ with the noun, so ”average“ *describes* ”Voice quality“. Using this reasoning, the following relations can be made out:

- The text is *madeUpOf* the sentence ”Voice quality was average.“
- The sentence ”Voice quality was average“ is *madeUpOf* the words Voice, quality, was and average.
- The words *isOfClass* as described in table 4.1
- The sentence is *madeUp* of the expressions ”Voice quality“ and ”was average“.
- The expression ”Voice quality“ *isA* noun phrase.
- The expression ”was average“ *isA* verb phrase.
- ”average“ *describes* ”voice quality“.

Figure 4.5 shows how these relations could be represented:

This information will be stored in the assertional component, a knowledge base, of the Text Ontology. The next section describes the ontology that will be used to identify descriptive terms.

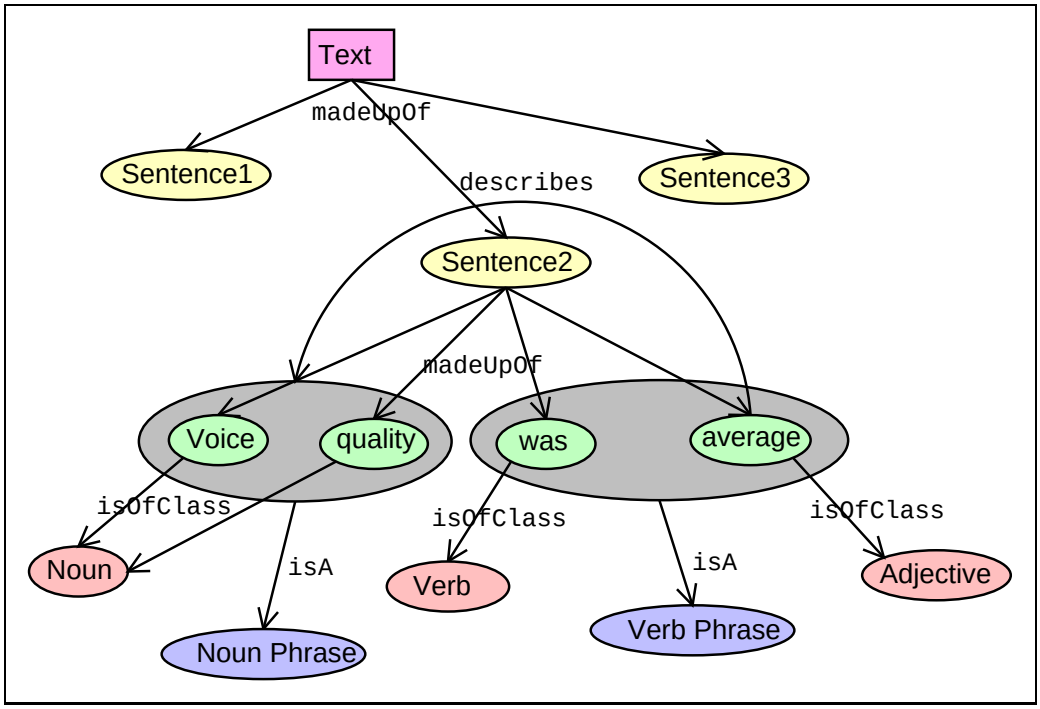


Figure 4.5: A schematic representation of some of the concepts and relations found in figure 4.1.

4.4 The Descriptions Ontology

The Descriptions Ontology uses the Word, Expression and Phrase concepts defined in the Text Ontology to create a lexicon of terms known to be descriptions. Considering this, the Descriptions Ontology uses these concepts:

- **Word** - From the Text Ontology.
- **Expression** - From the Text Ontology.
- **Phrase** - From the Text Ontology.
- **Descriptive Term** - A descriptive term is either a Word, an Expression or a Phrase.

4.5 The PreProcessor

If all authors took the time to make an ontological representation of all the articles or texts they wrote, they would make life for automated reasoners a lot easier. This is not the case, however, and the process of making such a representation of semi-structured

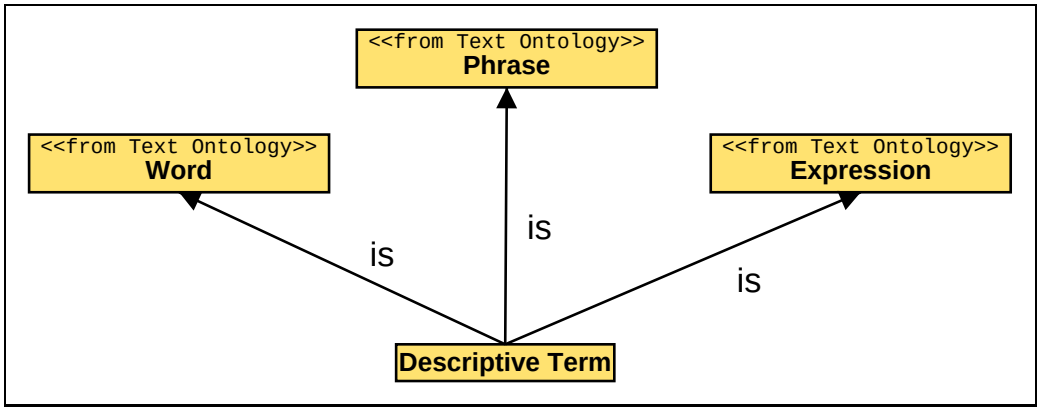


Figure 4.6: A schematic representation of the concepts and relations in the Descriptions Ontology.

text found in articles, books, reports, etc. will have to be done as a preprocess for a reasoner. This can be done by identifying as many of the relations between the words in the text as possible by natural language processing.

As most texts consist of more than one sentence, the first objective is to separate the text into a list of sentences. Then analysis of the individual sentences can start, the overall goal being to locate the subject and objects in the sentence and how they are described. The two main approaches to discover how an object is spoken of, are to locate any adjectives or expressions describing it, or the adverb(s) or expression describing the action the object is taking.

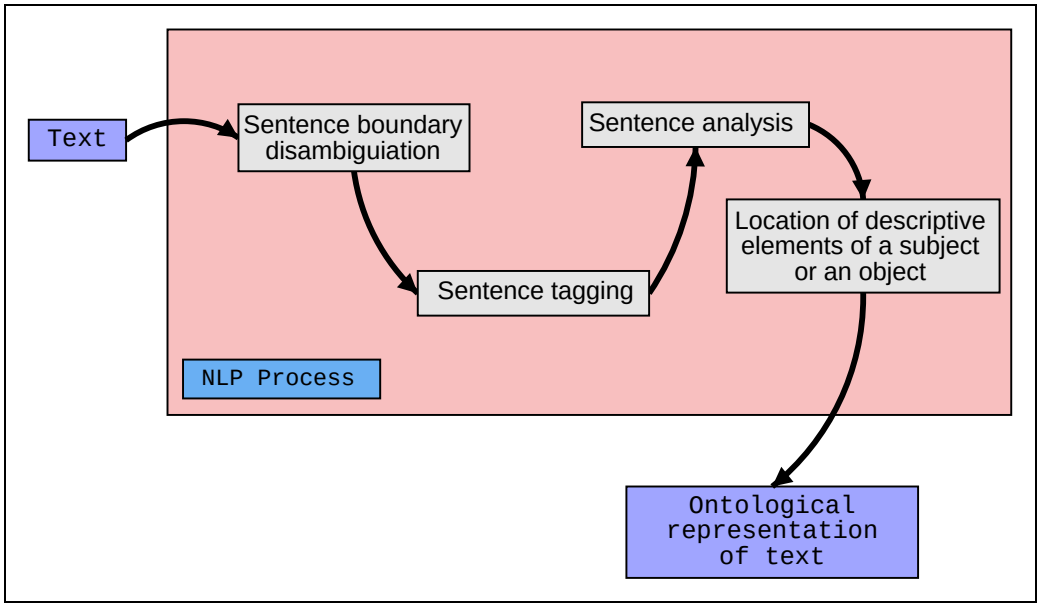


Figure 4.7: The steps involved in analyzing a text.

When this is done, the sentences, words, and interword relations are stored in an ontology knowledge base, as figure 4.7 illustrates.

Based on the description above, the PreProcessor will operate according to the following algorithm (listing 4.1):

- **Input**: *Text* (A natural language text)
- **Output**: An ontological representation of the text.

Listing 4.1: The PreProcessor Algorithm

```
# First discover the sentence boundaries.
sentences = sentence_boundary_disambiguation(Text)

# Tag the sentences with Part of Speech.
tagged_sentences = list
for sentence in sentences:
    tagged_sentence = pos_tag(sentence)
    tagged_sentences.append(tagged_sentence)

# Perform grammatical analysis.
sentence_parse_trees = list
for tsent in tagged_sentences:
    ptree = cfg_parse(tsent)
    sentence_parse_trees.append(ptree)

# Find description relations.
```

```
analyzed_sentences = list
for spt in sentence_parse_trees:
    analyzed_sentence = find_descriptions(spt)
    analyzed_sentences.append(analyzed_sentence).

# Generate ontological representation.
ontological_representation = represent_as_ontology(
    analyzed_sentences)

return ontological_representation
```

The implementation specifics for the functions in the algorithm can be found in section 6.3.

4.6 The Description Temperature Ontology

The Description Temperature Ontology will be used by the Reasoner to determine the temperature of how objects are spoken, hence this ontology must contain information about the temperature of descriptions; words and expressions.

4.6.1 The Terminological Component

An object description can be said to be warm (positive), cold (negative) or neutral. But in the real world, is it common to have a higher resolution when evaluating descriptions. They can be said to be very warm, for instance ”These are the perfect running shoes¡‘, while ”These running shoes are comfortable“ has not the same level of warmth in the description. Is it therefore necessary to develop a warm-cold scale. Words and expressions will then be placed along this scale when developing the Description Temperature Ontology. The different levels of warmth and cold used in the ontology are described in the following figure (figure 4.8):

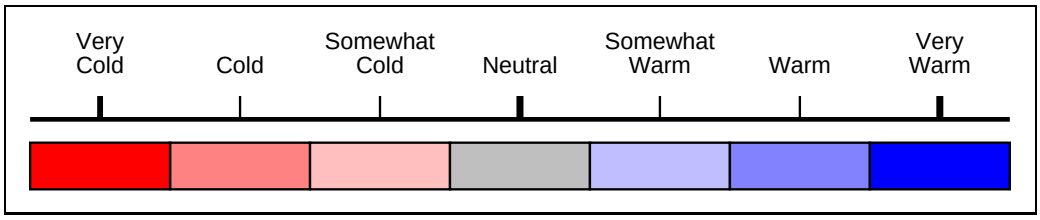


Figure 4.8: The Warm-Cold Scale used in the Description Temperature Ontology.

Considering this, the Description Temperature Ontology uses these concepts:

- **Descriptive Term** - From the Descriptions Ontology.

- **Temperature** - The level of warmth or cold in the description of an object.

The Descriptive Term is related to the Temperature concept by the *hasTemperature* relation. The following figure (figure 4.9) illustrates the concepts and relations:

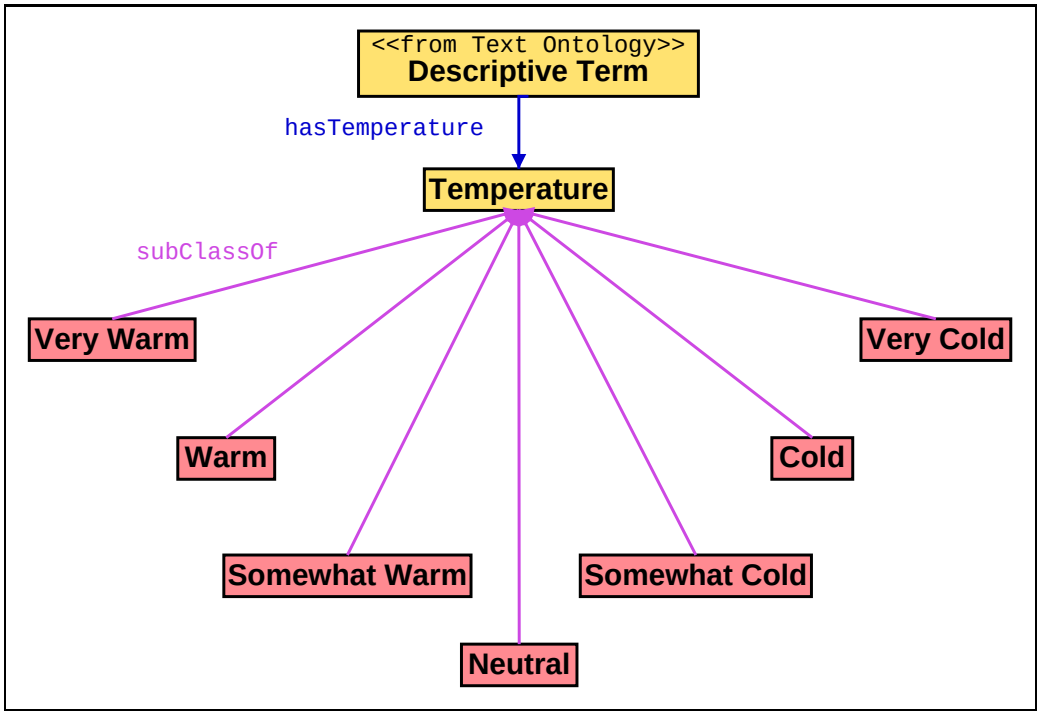


Figure 4.9: A schematic representation of the concepts and relations in the Description Temperature Ontology.

The *Temperature* concept will be applied to the elements found in the Descriptions Ontology. These definitions will work as classes when the assertional component of the Description Temperature Ontology is developed.

4.6.2 The Assertional Component

The assertional component of the Description Temperature Ontology will contain the temperature of descriptions found in the Descriptions Ontology. If a word or expression is absent from that ontology, a reasoner has no way of grading the temperature of a description. The following table contains an example of how expressions, words and phrases will be categorized (the terms are collected from the Descriptions Ontology);

TERM	TYPE	CATEGORY
acceptable	Word	Somewhat Warm
poor	Word	Cold
average	Word	Somewhat Cold
occasionally failing	Expression	Somewhat Cold

Table 4.2: An example of how phrases, expressions and words will be categorized in the Description Temperature Ontology using the descriptive terms from the example text in figure 4.1.

Section 6.5 describes how the ontology was implemented.

4.7 The Reasoner

The Reasoner is, as mentioned earlier, the processing unit that combines the Text Ontology the Descriptions Ontology and the Description Temperature Ontology to determine if a specific object or term is spoken of in a warm of cold manner (or neutral). This is done by combining a text processed by the PreProcessor (and hence is in the Text Ontology assertional component), with the warm-cold-graded terms in the Description Temperature Ontology.

4.7.1 Determining The Temperature of a Text Relative to a Term

To determine how an object is spoken, the sentences that describe the object is located. If any expressions, phrases or words *describes* the term in question, the temperature of this description determines how the object is spoken of. For example, consider ”Voice quality was average“ from earlier. The adjective ”average“ *describes* ”voice quality“. In the Description Temperature Ontology, the word average will typically be categorized as somewhat cold, hence voice quality in the sentence is described in a somewhat cold manner. The Reasoner identifies the words, expressions or phrases that describes the given object and looks these up in the Description Temperature Ontology. Figure 4.10 illustrates:

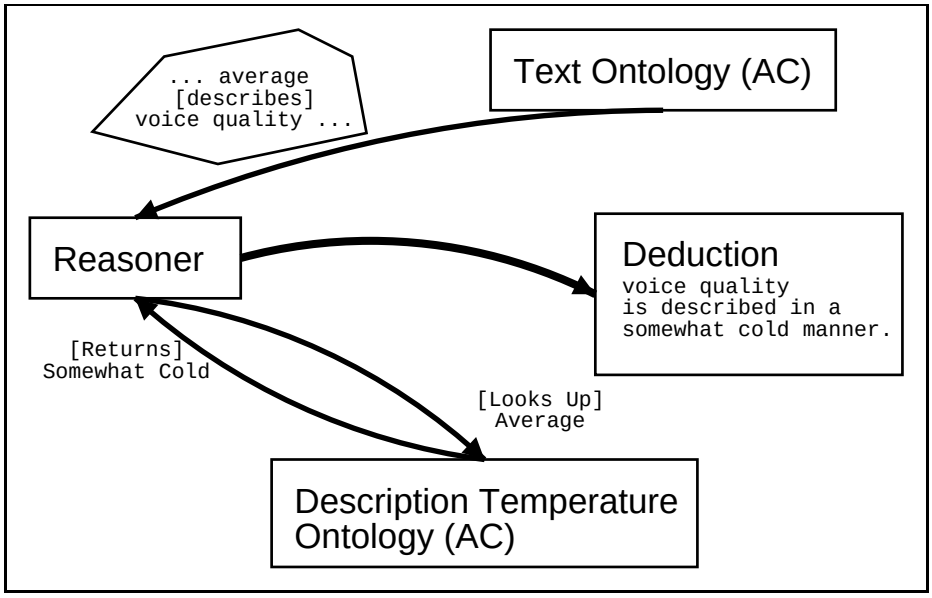


Figure 4.10: An example of how the Reasoner reasons about object descriptions.

Also, if the object in question is *theSameAs* another object that is described in the sentence, the Reasoner finds out how this object is spoken of. Knowing this, the algorithm of the Reasoner is as follows (listing 4.2):

- **Input**: *OntText* (A text from the Text Ontology.)
- **Input**: *Term* (The term that should be checked for warm or cold descriptions.)
- **Output**: The deduced conclusion of the general temperature of the text relative to the input term.

Listing 4.2: The Reasoner Algorithm

```
# Locate all sentences with the term in question.
sentences = locate_sentences_with_term(OntText, Term)

# Find all descriptions of the term.
descriptions = get_term_descriptions(senctences, Term)

# Get the temperatures of the descriptions.
desc_temps = get_description_temperature(descriptions)

# Compute a general impression of the temperature.
gen_temp = normalize_description_temperatures(desc_temps)

return gen_temp
```

The specifics for the implementation of the algorithm can be found in section 6.5.

Chapter 5

Identifying Shortcomings With the Proposed Method

As has become apparent while developing the proposed method, it suffers shortcomings that will become subject for further work. The sections below describes them in detail.

5.1 Going Beyond Single Sentence Analysis

When an object is referred to in several consecutive sentences, it is common to replace the object name with a personal pronoun to ease the reading. Consider this example (Example 5.1.1):

Example 5.1.1. *To summarize, P800 is an excellent cellular phone. It looks good as well.*

The proposed method does not consider such constructs (even though they are quite common). The word ”it“ and ”P800“ refers to the same object in these sentences, and it would be desirable to establish the *refersToTheSameAs* relation between these words to be able to locate more descriptions of the object. This way a more accurate conclusion could be made about how the object in question is spoken of. Typically - consider the example - the word ’P800’ would be used as few times as possible, and in the worst case scenarios, ’P800’ would have no relation to descriptive terms at all, only properties of the phone (signal strength, voice quality, etc.). In such cases, the proposed method breaks.

5.2 Going Beyond Single Text Analysis

Going beyond single sentence analysis will allow the identification of more descriptions. Going beyond single text analysis allows the identification of trends. Consider analyzing several texts from several sources in a given time span. Does any of these sources speak of my products more critically than the others? Less critically? Is the trend that the media speaks of our company more and more positive, or more and more negative? Have the resent marketing efforts made the online discussion groups talk more friendly of our company or our products? Such trends are time consuming to discover and to measure without the help of able automated agents.

Supplying enough meta data about the texts analyzed, it is possible to provide this and similar statistics automatically. This way we add the time dimension to the analysis of the temperatures of descriptions; *how has the "public opinion" of our company evolved through the last months?*

5.3 Identifying Interesting Terms Using Ontologies

The proposed method allows identification of how a term is spoken of in a text. However, imagine an automated marketing surveillance system that reports online articles that speak of Ericsson in some unfavorable fashion. Configuring the system, it is insufficient to report only when the word Ericsson is spoken of unfavorable. Ericsson produces cellular phones and other services, and negative reviews of these reflects badly upon the company itself.

Consider an Ericsson ontology, or even better, a *Telecom industry ontology*, mapping the Telecom industry landscape; all its actors, common terms and concepts, etc. (see figure 5.1) When configuring the system to report negative formulations about elements that concern Ericsson, the ontology will tell the reasoner that negative descriptions of the voice quality of the T68 mobile phone reflects negatively on the development department of Ericsson, and hence the company as a whole. Combining the current functionality of the proposed method with domain specific ontologies, as the Telecom Industry Ontology, would reduce configuration time and maintenance of the reasoner, in addition to obtaining an increase in areas of application by orders of magnitude.

5.4 Figures of Speech

There are a number of linguistic instruments available to make ones point when writing a text. Irony and sarcasm are such instruments. Both are frequently used in oral language, and some times in the written context. They are mostly used in causeries

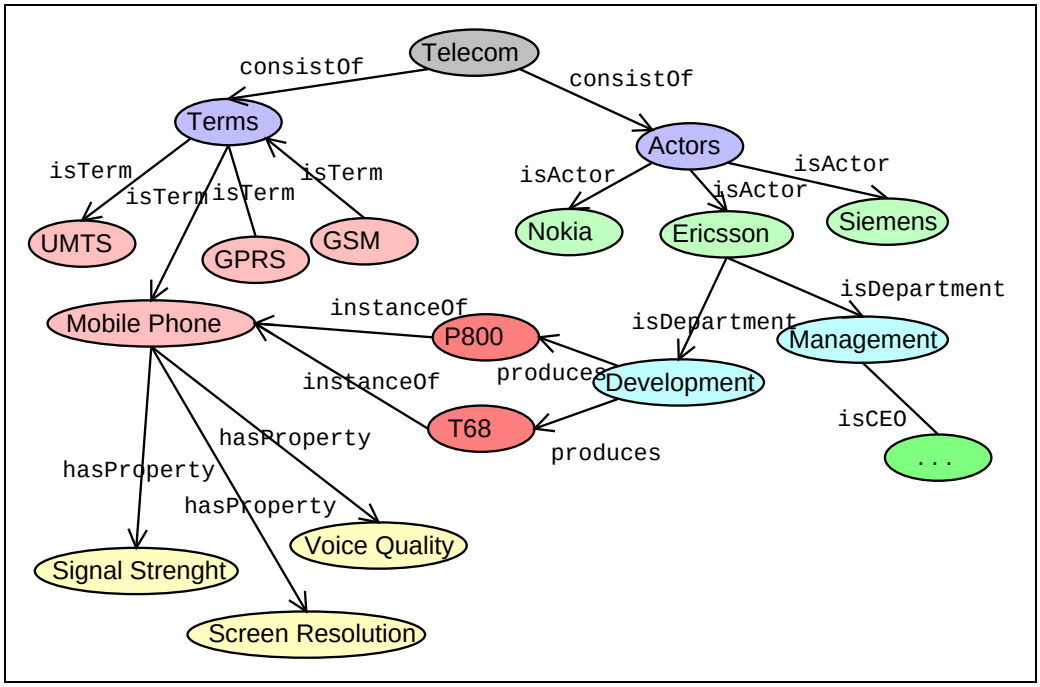


Figure 5.1: A Simplified Example of a Telecom Industry Ontology.

and contributions in newspapers and similar media.

Irony and sarcasm express the opposite of, or something different than, what is actually written. Sarcasm is more harsh than irony, and easier to realize.

Example 5.4.1. *That new haircut of yours is **so** nice...*

Some constructs of irony are more obvious than others, and people have different abilities to understand irony and sarcasm. It is both a matter of training and maturity. Children can not understand irony until they are ten years old, and scholars generally have an easier time understanding it than people scarcely-read.

Metaphors give words a figurative meaning.

Example 5.4.2. *You are an **angel***

Sometimes these can be extremely hard to perceive. ”House in the Dark“ by Tarjei Vesaas for instance, would be hard to make sense of without having reference to the second World War and Norway under the German occupation. It goes without saying that it is a challenging task to make a computer understand the meaning such metaphors; the same text could have two completely different meanings if it was written a decade sooner, or a decade later than the actual year of publication.

The writing techniques described above will ”confuse“ an implementation of the proposed solution, because it is not taken into consideration that an object can be described in any other way than directly, without the use of irony, sarcasm and similar.

Chapter 6

The Prototype

This section describes how the proposed approach is implemented. First, a description of the Text and Descriptions Ontologies are presented. Then, the PreProcessor and how it uses these two ontologies will be shown. Finally, the Description Temperatures Ontology and the Reasoner are implemented, followed by a presentation of the limitations of all aspects of the prototype.

6.1 The Terminological Component of the Text Ontology

All the ontologies developed were implemented using the Ontology Markup Language (OWL).[1] This section describes how the terminological component of the Text Ontology was implemented.

Based on the concepts and relations identified in section 4.3, the Text Ontology is implemented as follows (see *the OWL Semantics and Abstract Syntax* ([17]) for a detailed description of the OWL syntax). All the relations are mappings of those found on page 27.

Listing 6.1 defines the *Word* concept and the relations in the *Word* domain. The *wordText* property of the *Word* class is the textual representation of the word.

Listing 6.1: The Word Concept

```
<owl:Class rdf:ID="Word" />

<owl:DataTypeProperty rdf:ID="wordText">
   <rdfs:domain rdf:resource="#Word" />
```

[1] See the OWL Semantics and Abstract Syntax [17] and the OWL Guide [23] for a description of how to develop ontologies with OWL.

```
    <rdfs:range rdf:resource="http://www.w3c.org/2000/10/
        XMLSchema#string" />
</owl:DataTypeProperty>

<owl:ObjectProperty rdf:ID="isOfClass">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#PartOfSpeech" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="wordDescribesWord">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Word" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="wordDescribesExpression">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="wordIsTheSameAsWord">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Word" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="wordIsTheSameAsExpression">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>
```

Listing 6.2 defines the *Sentence* concept and the *sentenceMadeUpOfWord* relation.

Listing 6.2: The Sentence Concept

```
<owl:Class rdf:ID="Sentence" />

<owl:ObjectProperty rdf:ID="sentenceMadeUpOfWord">
    <rdfs:domain rdf:resource="#Sentence" />
    <rdfs:range rdf:resource="#Word" />
</owl:ObjectProperty>
```

Listing 6.3 defines the *Phrase* concept and the relations it its domain.

Listing 6.3: The Phrase Concept

```
<owl:Class rdf:ID="Phrase" />

<owl:ObjectProperty rdf:ID="isOfType">
    <rdfs:domain rdf:resource="#Phrase" />
    <rdfs:range rdf:resource="#PharaseType" />
```

```
</owl:ObjectProperty>
```

Listing 6.4 defines the *Text* concept and the relations it its domain.

Listing 6.4: The Text Concept

```
<owl:Class rdf:ID="Text" />

<owl:ObjectProperty rdf:ID="madeUpOfSentence">
   <rdfs:domain rdf:resource="#Text" />
   <rdfs:range rdf:resource="#Sentence" />
</owl:ObjectProperty>
```

Listing 6.5 defines the *Expression* concept and the relations it its domain.

Listing 6.5: The Expression Concept

```
<owl:Class rdf:ID="Expression" />

<owl:ObjectProperty rdf:ID="expressionMadeUpOfWord">
   <rdfs:domain rdf:resource="#Expression" />
   <rdfs:range rdf:resource="#Word" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="expressionCanBeA">
   <rdfs:domain rdf:resource="#Expression" />
   <rdfs:range rdf:resource="#Phrase" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="expressionDescribesWord">
   <rdfs:domain rdf:resource="#Expression" />
   <rdfs:range rdf:resource="#Word" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="expressionDescribesExpression">
   <rdfs:domain rdf:resource="#Expression" />
   <rdfs:range rdf:resource="#Expression" />
</owl:ObjectProperty>
```

Listing 6.6 defines the *PartOfSpeech* concept and the relations it its domain. It also defines the instances of the *PartOfSpeech* found in the Penn Treebank Tag Set (see Appendix B).

Listing 6.6: The PartOfSpeech Concept

```
<owl:Class rdf:ID="PartOfSpeech">
   <rdfs:label xml:lang="en">Part of Speech</rdfs:label>
</owl:Class>
```

```
<PartOfSpeech rdf:ID="CC" />
<PartOfSpeech rdf:ID="CD" />
<PartOfSpeech rdf:ID="DT" />
<!-- Entire list omitted. -->

<owl:AllDifferent>
   <owl:distinctMembers rdf:parseType="Collection">
      <text:PartOfSpeech rdf:about="#CC" />
      <text:PartOfSpeech rdf:about="#CD" />
      <text:PartOfSpeech rdf:about="#DT" />
      <!-- Entire list omitted. -->
   </owl:distinctMembers>
</owl:AllDifferent>
```

Listing 6.7: The PhraseType Concept

```
<owl:Class rdf:ID="PhraseType" />

<PhraseType rdf:ID="NP" />
<PhraseType rdf:ID="VP" />
<PhraseType rdf:ID="PP" />

<owl:AllDifferent>
   <owl:distinctMembers rdf:parseType="Collection">
      <text:PhraseType rdf:about="#NP" />
      <text:PhraseType rdf:about="#VP" />
      <text:PhraseType rdf:about="#PP" />
      <!-- Entire list omitted. -->
   </owl:distinctMembers>
</owl:AllDifferent>
```

See Appendix C.1 for the entire OWL implementation of the terminological component.

6.2 The Descriptions Ontology

The Descriptions Ontology is of the simplest kinds of ontologies; lexicons. It's just a list of terms and what kind of term it is. Most of the concepts required to implement the ontology is defined in the Text Ontology. The Descriptions Ontology only defines the term *Descriptive Term*.

Note 6.1. It is implied that all adjectives and adverbs are descriptive terms.

6.2.1 Terminological Component

The terminological component defines the *DescriptiveTerm* concept used to point out that a term is descriptive about an object.

Listing 6.8: The Description Concept

```
<owl:Class rdf:ID="DescriptiveTerm" />
<owl:DataTypeProperty rdf:ID="term">
   <rdfs:domain rdf:resource="#DescriptiveTerm" />
   <rdfs:range rdf:resource="./text#Word" />
   <rdfs:range rdf:resource="./text#Expression" />
   <rdfs:range rdf:resource="./text#Phrase" />
</owl:DataTypeProperty>
```

6.2.2 Assertional Component

The assertional component of the Descriptions Ontology defines terms that are considered descriptive. Following is an excerpt from the ontology, defining the descriptive terms from the example text from figure 4.1:

Listing 6.9: Some descriptive terms.

```
<Word rdf:ID="acceptable">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">acceptable</wordText>
</Word>
<Word rdf:ID="poor">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">poor</wordText>
</Word>
<Word rdf:ID="average">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">average</wordText>
</Word>
<Word rdf:ID="occasionally">
```

```
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">occasionally</wordText>
</Word>
<Word rdf:ID="failing">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">failing</wordText>
</Word>
<Expression rdf:ID="cf4633f65058a0fe5f323c75edd8e2e5">
    <expressionMadeUpOfWord rdf:resource="#occasionally" />
    <expressionMadeUpOfWord rdf:resource="#failing" />
</Expression>

<DescriptiveTerm rdfID="desc_acceptable">
    <term rdf:resource="acceptable">
</DescriptiveTerm>

<DescriptiveTerm rdfID="desc_poor">
    <term rdf:resource="poor">
</DescriptiveTerm>

<DescriptiveTerm rdfID="desc_average">
    <term rdf:resource="average">
</DescriptiveTerm>

<DescriptiveTerm rdfID="desc_cf4633f65058a0fe5f323c75edd8e2e5"
    >
    <term rdf:resource="cf4633f65058a0fe5f323c75edd8e2e5">
</DescriptiveTerm>
```

See Appendix C.1 for the entire OWL implementation of the Descriptions Ontology.

6.3 The PreProsessor

The PreProcessor takes a text as input and outputs an ontological representation of the same text. This process includes separating the text into sentences, tagging the text, giving reasonable parses of the text, and use the OWL implementation of the *Text Ontology* to get a OWL representation of the text. The PreProcessor is run using Jython 2.1.

The functionality of the PreProcessor has been wrapped in two Python (Jython) components;

- **TextAnalyzer** - This class wraps functions to find sentence boundaries (using the MXTERMINATOR classes [24]), tags the sentences with part of speech (using the Monty Tagger [25]) and performs sentence analysis to discover phrases

in the text (using the Python Natural Language ToolKit [26]). Output from the TextAnalyser.getPhrases function is used to get an OWL representation of the text using an object of the OWLRepresentator class.

- **OWLRepresentator** - This class has functions to represent a sentence parse-tree as an OWL ontology. It also generates relations between words beyond that of the TextAnalyser, discovering what describes what in a text. This class is used to develop the assertional component of the Text Ontology, and assigns IDs to texts using the MD5-algorithm.

6.3.1 TextAnalyser

As described above, the TextAnalyser performs most natural language processing involved the system.

To divide the text into sentences, the analyzer utilizes an instance of the class Text_processor. This is a wrapper of the MXPOST and MXTERMINATOR described in section A.1.5. The Text_processor is written in Java. It has a function called *sentence_boundary*, which takes the text to be processed as an argument and returns a string divided by '\n' (line break) as sentences. MXTERMINATOR writes all its results to System.out (to circumvent this, i.e. allowing the use of the MXTERMINATOR outside the command line, System.out is redirected to a String buffer, instead of to the console).

For all sentences there exist a common set of CFGProduction objects(2.5.2) that contains context free grammar rules to identify phrases in a sentence (e.g. noun phrase, verb phrase etc.) See table 6.1 on page 49 for an example of how these grammars are built. They are combined with rules generated by the Brill-tagger - the tuple of word and tag for each word in the sentence (see section A.1.6 for an explanation of why this POS tagger has been chosen for the implementation). The sum of all these rules are used as parameter to a NLTKParser object, which uses them to generate a Chart object that describes the sentence as a parse-tree. Figure 6.1 illustrates;

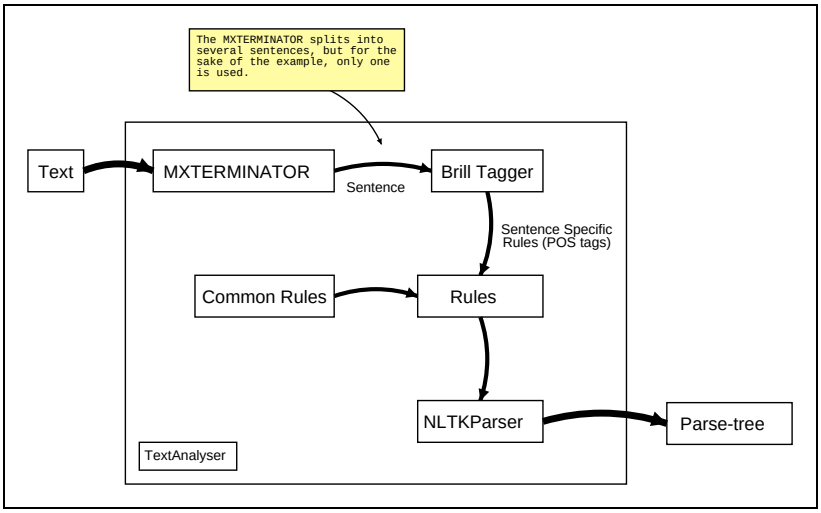


Figure 6.1: The TextAnlyser components.

Figure 6.2 shows an UML diagram of the TextAnalyzer class, which has a method called getPhrases. This method performs all the natural language processing described above.

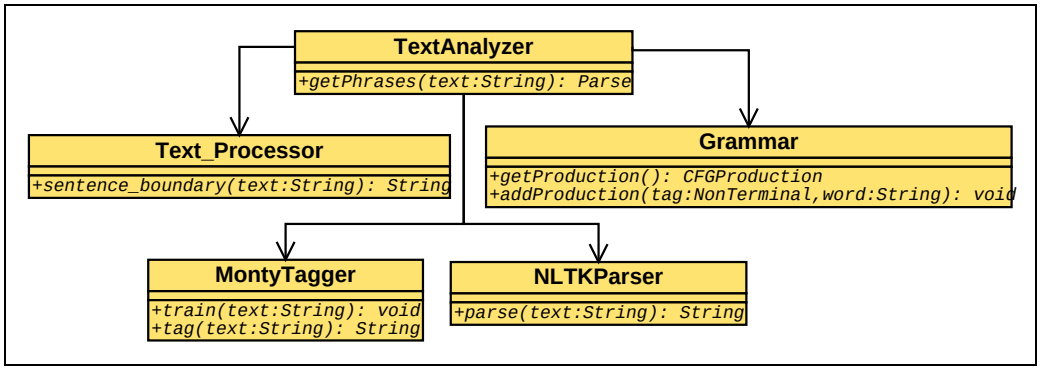


Figure 6.2: UML-diagram of the TextAnalyzer component.

Example Run

The following figure (figure 6.3) illustrates how the text from figure 4.1 is affected by the different steps involved in the processing. For the sake of the example, when the text has been split into sentences, only one of them (the first) is used to explain how grammar rules and parse-trees are generated. The left side of the dotted line is the component the text is being processed by, and the right side is the text after the processing. The sentence parse-trees are used as parameter the the OWLRepresentator described in the next section.

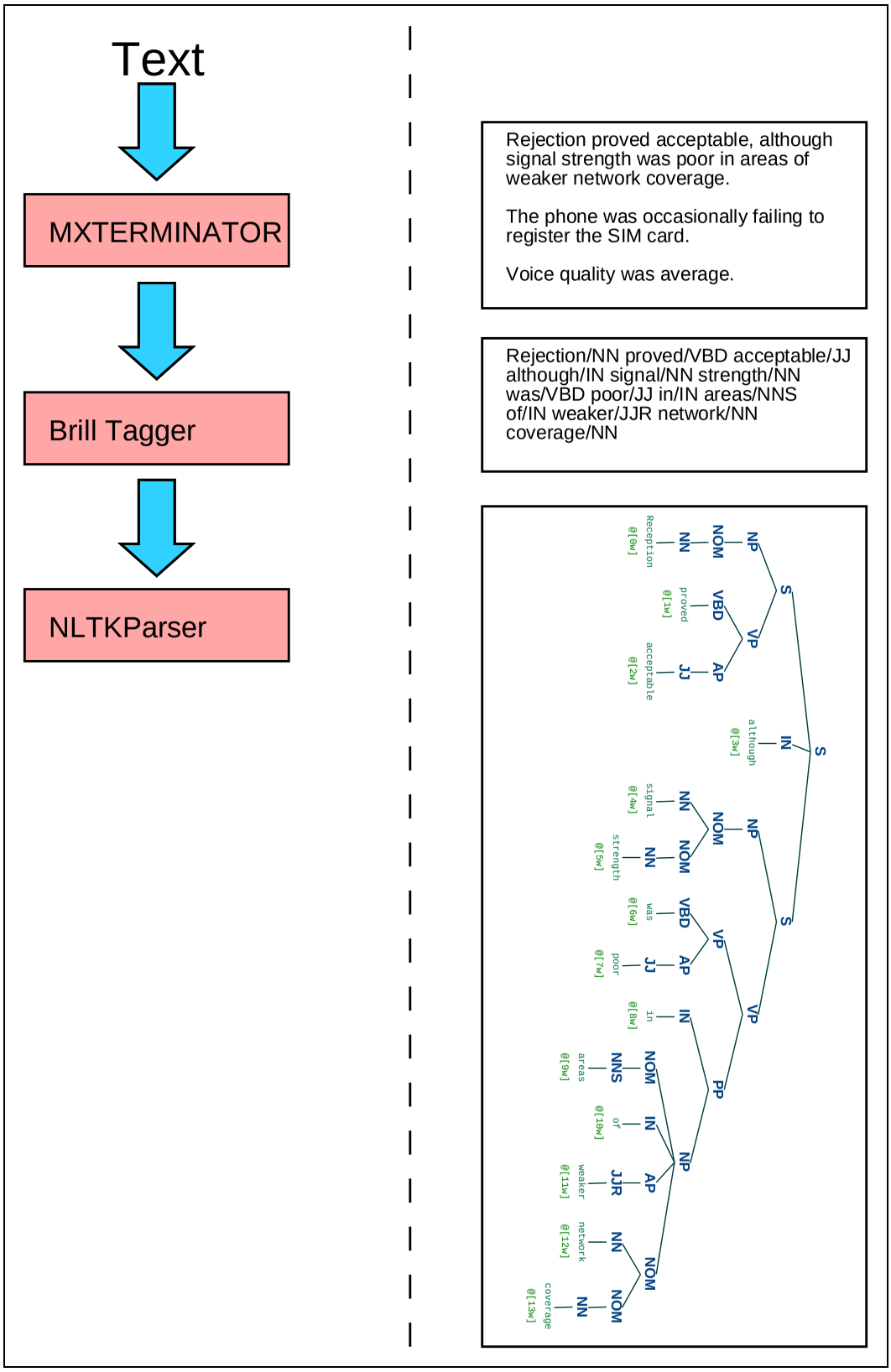


Figure 6.3: An example of the processing performed by the TextAnalyzer class.

ROOT		RULE	DESCRIPTION
S	→	***NP VP***	A sentence consists of a noun phrase and a verb phrase.
S	→	***S IN S***	A sentence may be to complite sentences divided by a conjunction.
NP	→	***NP NOM***	A nominal is a noun phrase.
NP	→	***PRP***	A pronoun is a noun phrase.
VP	→	***VP NP***	A verb phrase followed by a noun phrase.
VP	→	***VP PP***	A verb phrase followed by a prepositional phrase.
VP	→	***VBD AP***	A verb (past tense) followed by a adjective phrase.
VP	→	***VBD AP***	A verb in past tense followed by a adjectiv phrase.
PP	→	***IN NP***	A sentence consists of a noun phrase and a verb phrase.
PP	→	***TO NP***	A sentence consists of a noun phrase and a verb phrase.
NOM	→	***NN***	A noun is a nominal.
NOM	→	***NN NOM***	A noun followed by a nominal.
AP	→	***JJ***	A adjective phrase is an adjective.

Table 6.1: A sample grammar.

6.3.2 OWLRepresentator

The OWLRepresentator is a Python class used to represent a list of sentence parse trees as OWL. This class also has the responsibility to generate the relations *describes* and *referToTheSameAs* between terms. It generates instances of the concepts defined in the Text Ontology based on the information in the parse trees. These instances make up the assertional component of the Text Ontology.

To be able to identify descriptive terms, the OWLRepresentator uses the Descriptions Ontology lexicon (as mentioned earlier, all adjectives and adverbs are descriptions per definition). The entire Descriptions Ontology is loaded on initialization and is wrapped in the Descriptions class. The following UML model describes the classes and relations between them. The two locator classes uses simple heuristics to locate the *describes* and *refersToTheSameAs* relations.

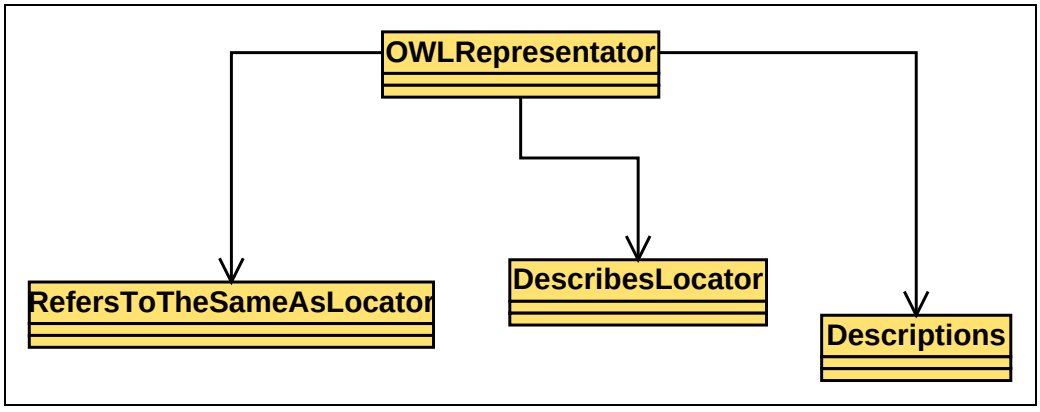


Figure 6.4: UML-diagram of the OWLRepresentator component.

The Assertional Component of the Text Ontology

The OWLRepresentator generates OWL from the output from the TextAnalyzer. Continuing the example in figure 6.3, inputing the parse trees returned from the TextAnalyzer yields:

Listing 6.10: The output OWL from the parse trees in figure 6.3.

```
<Word rdf:ID="acceptable_JJ">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">acceptable</wordText>
   <isOfClass rdf:resource="#JJ" />
   <describes rdf:resource="#Reception_NN" />
</Word>
<Word rdf:ID="poor_JJ">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">poor</wordText>
   <isOfClass rdf:resource="#JJ" />
   <describesExpression rdf:resource="#
      uhjdffu8brs743cbv7c9sdlxsf2cera6je">
</Word>
<Word rdf:ID="SIM_NNP">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">SIM</wordText>
   <isOfClass rdf:resource="#NNP" />
</Word>
<Word rdf:ID="average_JJ">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">average</wordText>
   <isOfClass rdf:resource="#JJ" />
   <describesExpression rdf:resource="#
      df4dfgf898brs743c57c9sd8f2c68a678f">
</Word>
<Word rdf:ID="although_IN">
```

```
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">although</wordText>
    <isOfClass rdf:resource="#IN" />
</Word>
<Word rdf:ID="phone_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">phone</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="failing_VBG">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">failing</wordText>
    <isOfClass rdf:resource="#VBG" />
</Word>
<Word rdf:ID="proved_VBD">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">proved</wordText>
    <isOfClass rdf:resource="#VBD" />
</Word>
<Word rdf:ID="strength_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">strength</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="The_DT">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">The</wordText>
    <isOfClass rdf:resource="#DT" />
</Word>
<Word rdf:ID="was_VBD">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">was</wordText>
    <isOfClass rdf:resource="#VBD" />
</Word>
<Word rdf:ID="Reception_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">Reception</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="the_DT">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">the</wordText>
    <isOfClass rdf:resource="#DT" />
</Word>
<Word rdf:ID="of_IN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">of</wordText>
```

```
    <isOfClass rdf:resource="#IN" />
</Word>
<Word rdf:ID="areas_NNS">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">areas</wordText>
    <isOfClass rdf:resource="#NNS" />
</Word>
<Word rdf:ID="signal_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">signal</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="weaker_JJR">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">weaker</wordText>
    <isOfClass rdf:resource="#JJR" />
</Word>
<Word rdf:ID="Voice_NNP">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">Voice</wordText>
    <isOfClass rdf:resource="#NNP" />
</Word>
<Word rdf:ID="to_TO">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">to</wordText>
    <isOfClass rdf:resource="#TO" />
</Word>
<Word rdf:ID="card_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">card</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="register_VB">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">register</wordText>
    <isOfClass rdf:resource="#VB" />
</Word>
<Word rdf:ID="network_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">network</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="coverage_NN">
    <wordText rdf:datatype="http://www.w3c.org/2000/10/
        XMLSchema#string">coverage</wordText>
    <isOfClass rdf:resource="#NN" />
</Word>
```

```
<Word rdf:ID="quality_NN">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">quality</wordText>
   <isOfClass rdf:resource="#NN" />
</Word>
<Word rdf:ID="in_IN">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">in</wordText>
   <isOfClass rdf:resource="#IN" />
</Word>
<Word rdf:ID="occasionally_RB">
   <wordText rdf:datatype="http://www.w3c.org/2000/10/
      XMLSchema#string">occasionally</wordText>
   <isOfClass rdf:resource="#RB" />
</Word>
<Sentence rdf:ID="c96deb4397a0ea4ae6fc24f401bcfcf7">
   <sentenceMadeUpOfWord rdf:resource="#Reception_NN" />
   <sentenceMadeUpOfWord rdf:resource="#proved_VBD" />
   <sentenceMadeUpOfWord rdf:resource="#acceptable_JJ" />
   <sentenceMadeUpOfWord rdf:resource="#although_IN" />
   <sentenceMadeUpOfWord rdf:resource="#signal_NN" />
   <sentenceMadeUpOfWord rdf:resource="#strength_NN" />
   <sentenceMadeUpOfWord rdf:resource="#was_VBD" />
   <sentenceMadeUpOfWord rdf:resource="#poor_JJ" />
   <sentenceMadeUpOfWord rdf:resource="#in_IN" />
   <sentenceMadeUpOfWord rdf:resource="#areas_NNS" />
   <sentenceMadeUpOfWord rdf:resource="#of_IN" />
   <sentenceMadeUpOfWord rdf:resource="#weaker_JJR" />
   <sentenceMadeUpOfWord rdf:resource="#network_NN" />
   <sentenceMadeUpOfWord rdf:resource="#coverage_NN" />
</Sentence>
<Sentence rdf:ID="e854aff8f55b016058e7eec46974729d">
   <sentenceMadeUpOfWord rdf:resource="#The_DT" />
   <sentenceMadeUpOfWord rdf:resource="#phone_NN" />
   <sentenceMadeUpOfWord rdf:resource="#was_VBD" />
   <sentenceMadeUpOfWord rdf:resource="#occasionally_RB" />
   <sentenceMadeUpOfWord rdf:resource="#failing_VBG" />
   <sentenceMadeUpOfWord rdf:resource="#to_TO" />
   <sentenceMadeUpOfWord rdf:resource="#register_VB" />
   <sentenceMadeUpOfWord rdf:resource="#the_DT" />
   <sentenceMadeUpOfWord rdf:resource="#SIM_NNP" />
   <sentenceMadeUpOfWord rdf:resource="#card_NN" />
</Sentence>
<Sentence rdf:ID="e63ede55066a9041b086fcd30cd34fb6">
   <sentenceMadeUpOfWord rdf:resource="#Voice_NNP" />
   <sentenceMadeUpOfWord rdf:resource="#quality_NN" />
   <sentenceMadeUpOfWord rdf:resource="#was_VBD" />
```

```
   <sentenceMadeUpOfWord rdf:resource="#average_JJ" />
</Sentence>
<Text rdf:ID="4f898b743c57c98f2c68a678f43edfc9">
   <madeUpOfSentence rdf:resource="#
      c96deb4397a0ea4ae6fc24f401bcfcf7">
   <madeUpOfSentence rdf:resource="#
      e854aff8f55b016058e7eec46974729d">
   <madeUpOfSentence rdf:resource="#
      e63ede55066a9041b086fcd30cd34fb6">
</Text>
<Expression rdf:ID="df4dfgf898brs743c57c9sd8f2c68a678f">
   <expressionMadeUpOfWord rdf:resource="#signal_NN" />
   <expressionMadeUpOfWord rdf:resource="#strength_NN" />
</Expression>
<Expression rdf:ID="uhjdffu8brs743cbv7c9sdlxsf2cera6je">
   <expressionMadeUpOfWord rdf:resource="#Voice_NN" />
   <expressionMadeUpOfWord rdf:resource="#quality_NN" />
</Expression>
```

6.4 The Description Temperature Ontology

The Description Temperature Ontology defines the terms identified in section 4.6. It also contains the *Temperature* instances from figure 4.8. These instances are related to descriptive terms in the Descriptions Ontology to set their temperature.

6.4.1 Terminological Component

The Description Temperature Ontology defines the *Temperature* and the *hasTemperature* relation. In order to relate a temperature to a descriptive term, the concept *TemperatedDescription* is also defined. The relation *temperates* relates a *DescriptiveTerm* and a *TemperatedDescription*. The following listings shows how this is implemented:

Listing 6.11: The Temperature Concept and the Temperature instances.

```
<owl:Class rdf:ID="Temperature" />

<Temperature rdf:ID="VeryCold" />
<Temperature rdf:ID="Cold" />
<Temperature rdf:ID="SomewhatCold" />
<Temperature rdf:ID="Neutral" />
<Temperature rdf:ID="SomewhatWarm" />
<Temperature rdf:ID="Warm" />
<Temperature rdf:ID="VeryVarm" />
```

Listing 6.12: The TemperatedDescription Concept.

```
<owl:Class rdf:ID="TemperatedDescription" />
<owl:ObjectProperty rdf:ID="temperates">
   <rdfs:domain rdf:resource="#TemperatedDescription" />
   <rdfs:range rdf:resource="#DescriptiveTerm" />
</owl:ObjectProptery>
<owl:ObjectProperty rdf:ID="hasTemperature">
   <rdfs:domain rdf:resource="#TemperatedDescription" />
   <rdfs:range rdf:resource="#Temperature" />
</owl:ObjectProperty>
```

6.4.2 Assertional Component

The asseritonal component of the Description Temperature Ontology relates a *Temperature* instance to a *DescriptiveTerm* through the *TemperatedTerm*. The following listing (listing 6.13) assigns temperatures to the descriptive terms defined in listing 6.9.

Listing 6.13: The assigned temperatures to the descriptive terms in listing 6.9.

```
<TemperatedDescription rdf:ID="temp_desc_acceptable">
   <temperates rdf:resource="#desc_acceptable" />
   <hasTemperature rdf:resource="#SomewhatWarm" />
</TemperatedDescription>
<TemperatedDescription rdf:ID="temp_desc_poor">
   <temperates rdf:resource="#desc_acceptable" />
   <hasTemperature rdf:resource="#Cold" />
</TemperatedDescription>
<TemperatedDescription rdf:ID="temp_desc_average">
   <temperates rdf:resource="#desc_acceptable" />
   <hasTemperature rdf:resource="#SomewhatCold" />
</TemperatedDescription>
<TemperatedDescription rdf:ID="
   temp_desc_cf4633f65058a0fe5f323c75edd8e2e5">
   <temperates rdf:resource="#
      desc_cf4633f65058a0fe5f323c75edd8e2e5" />
   <hasTemperature rdf:resource="#SomewhatCold" />
</TemperatedDescription>
```

See Appendix C.3 for the entire OWL implementation of the Description Temperature Ontology.

6.5 The Reasoner

The Reasoner loads a *Text* instance from the Text Ontology, locates all expressions describing the desired term and checks the temperatures of these expressions. The coldest temperature describing the object in question is returned as the final conclusion.

Note 6.2. No normalization of the temperatures of the descriptive expressions is performed. This is due to the ”warning-flag“ perspective of the implementation of the Reasoner; *”Let me know if anybody is discontent with . . . “*.

The Reasoner is wrapped in a Python (Jython) application, the main functionality of which being accessed through the Reasoner class. The UML diagram in figure 6.5 shows the design:

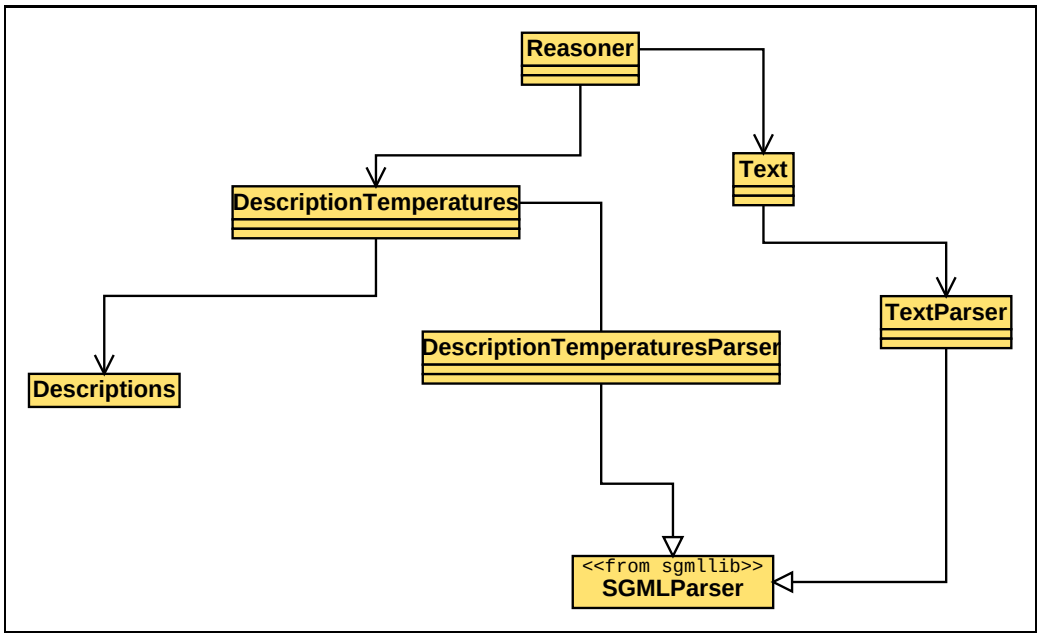


Figure 6.5: UML-diagram of the Reasoner component.

The text OWL representation is wrapped in the Text class and allows extraction of expressions that describe a given term. All the descriptions and their temperatures are accessed through the DescriptionTemperatures class. Both the Text and DescriptionTemperatures classes use a Python sgmllib.SGMLParser to process the OWL ontologies.

6.6 Limitations

There are several aspects about the prototype that limits its usage. Citing from the thesis definition (section 1.3), *A prototype that demonstrates how it works will also be*

developed, nutshells these limitations; natural language processing is a tricky business. It was never the intention, nor would it have been a realistic one, to be able to process all constructs of the English language. The same goes for establishing relations between descriptions and objects. To be able to do this in a general and fail proof way would require research beyond the scope of the thesis definition. The intention with the prototype was to prove that the proposed solution to identify how objects are spoken of in natural language texts, is one way to approach the problem. However, the main aspects of the prototype that compromises the degree of correctness of the conclusions are listed below:

- **Sentence Boundary** - The sentence boundary disambiguation algorithm is based on an abbreviation list. Abbreviations absent from this list may result in incorrectly placed sentence breaks. This may in turn compromise the entire text analysis.
- **Tagging** - None of the POS taggers tested were 100% accurate. Inaccuracy in the part of speech tagging has consequences for the grammatical analysis and the OWL representation of the text.
- **Parsing** - The sentence parsing module requires every aspect of the grammars in the language analyzed defined. Undefined constructs may result in the parser failing to assign phrases to the different parts of the sentence all together. If this happens, the program will break.
- **Performance** - In the prototype, the performance has been given a lower priority, hence the speed of the system is low. The weakest link of performance is the text parser, and it will continue to get slower as the amount of rules increases. The amount of rules will have to increase through time to map to the entire English grammar.
- **Sentence Parts** - A sentence may contain several smaller independent or dependent parts.
  - A sentence could be made from a subordinate clause, before the main clause.

    Example 6.6.1. *One problem we faced was the phone occasionally failed to register the SIM card.*
  - A sentence may contain a reference to something that has been said.

    Example 6.6.2. *The production manager of Ericsson said:"The P800 will set a new sales record".*

  The different parts are usually divided by comma followed by a conjunction. Such constructs are not supported in the set of grammatical rules of the prototype.

- **Representing Relations** - The set of rules used to identify what makes a phrase describe another is limited, and is missing a number grammatical constructs.
- **Ontologies** - The ontologies developed to represent descriptions and their temperatures are in no way complete. Developing such ontologies is a continual and time consuming process spanning several years.

Chapter 7

Discussion

Before being able to implement the proposed method, several implementation issued had to be resolved. The project started with a thorough literature study on both computational natural language processing, and ontology technologies, both of which we had scarce or no previous knowledge. Several different approaches to sentence disambiguation, part of speech tagging, grammatical analysis and ontology representation were covered. This was necessary to be able to make sensible identifications of the tools required to complete the task at hand.

The purpose of the project was to identify how ontologies combined with natural language processing could aid an automated agent in identifying how objects are spoken of in texts.

We have divided the topics of the discussion into the following groups:

- The Natural Language Processing
- The Representation of Text
- The Description Temperatures
- The Prototype
- Further Work

7.1 The Natural Language Processing

After having read up on the basics of natural language processing, we had to locate and decide what tools to use to obtain our goals. As mentioned in the introduction, the criterion was that the tools were freely available, preferably under a GPL license. There proved to be a number of these available with different capabilities (the ones we tested, and a thorough description of them can be found in chapter A).

We were aware of two toolkits for text processing from earlier; the Python Natural Language Toolkit (NLTK) and the Grok toolkit, but we had no experience with any of them. The Grok-toolkit (for Java) was too complicated to get control of quickly, while the NLTK-toolkit, providing the same or adequate functionality, was much easier to use. We also preferred Python as our prototype implementation language, so the selection process was easy. An other reason for choosing NLTK, was that Grok had its release of version 0.7, the first final release, finished at 24th of February this winter.

Decision about choosing a part of speech tagger was a bit harder. Since we used Python as our language of implementation, we had available taggers written in Java as well, through Jython. We performed a test of four different taggers (the results of this test can be found in Appendix A), ending up with the most accurate one, the Brill based Monty Tagger, disregarding performance issues.

Throughout the development process we have had accuracy as a number one priority, considering speed only in cases where the tool spent extreme amounts of time (this did not happen with any of the tools in the test). Having speed as a secondary priority, however, has had its consequences. Running the Monty Tagger and NLTK through Jython is very inefficient. Jython is implemented in Java, and is therefore slower than the original C implementation of Python. The Monty Tagger is originally a Java implementation and is hence only available to us through Jython (this is what stuck us to the Jython platform, as we wanted to use NLTK). NLTK tree generation also runs slower on the Jython platform than on the C implementation. The following table presents performance comparisons of running the different toolkits on the Java, Jython and Python platforms, where available, using the text from figure 4.1 (initialization of the toolkit in question is included the timing). As can be seen from the table, the performance of

Platform / Toolkit	**Java**	**Jython**	**Python**
Monty Tagger	0.51	0.53	N/A
NLTK Grammars	N/A	213 s	88 s

Table 7.1: Performance in seconds.

Jython is considerably lower than that of Java and Python. Hence, utilizing Jython to combine the desired tools in a seamless manner, compromises performance compared with that feasible when running the tools in their optimal environments.

Another thing affecting the execution speed, is the number of rules defined when generating the grammar trees. The grammar defined in our thesis is only a subset of the grammar of the English language, but still the generation of grammar trees is the most time consuming of the processes the texts go through. Platforms and other implementation specifics aside, grammatical analysis is a time consuming process, because the

generation of grammar trees is recursive - a verb phrase can contain a verb phrase and a noun phrase, for instance. It is not realistic to create specific grammatical rules for every possible construct of a sentence, compromising performance.

7.2 The Representation of Text

The representation of the ontologies we developed during this project is done with the Web Ontology Language specified by the World Wide Web Consortium. Having decided to do this already before the project started (because we wanted to use a format that was a standard), this also had its negative consequences. OWL is still a work in progress. Not lacking documentation, though, the OWL tool box is still missing the most useful libraries. As OWL is based on the more established DAML+OIL, and DAML+OIL would probably be adequate for our purposes, it might be argued that we should have chosen this ontology mark up language instead. However, writing the OWL parsers based on the Python sgmllib.SGMLParser, was not of the most time consuming tasks, and this way we learned the OWL language better. We still feel confident that the selection was the right one.

Establishing the *describes* relation was hard to make work in all cases. The rules for doing this are mostly based on heuristics from discussion groups and own ideas. This should probably be developed into a bundle of formal rules.

7.3 The Description Temperatures

”Temperating“ descriptive terms circumvents the Comparative Expression Ambiguity Problem [27] because assigning temperatures to a descriptive term is not the same as assigning meaning. Saying that some object A is *better than* some other object B, and temperating *better than* as *Warm,* does not say anything about what it means to be better, only that being better is probably positive. This way, the temperature of a description is an implied property of the description, and could hence be deduced from an ontology where terms had assigned meanings. Having this kind of ontologies available, would automate the process of building the ontologies required to make our prototype functional.

7.4 The Prototype

As implied in the introduction and thesis definition, the prototype is of the Proof-of-Concept kind. We have shown that it is possible to implement the proposed approach, and in the process of developing both the methods and prototype, we have come across

several shortcomings that we feel incomplete our work and are subject for future enhancements. These are presented in section 7.5.

As mentioned in section 6.6, only a subset of the English grammar has been included in the prototype. Also, the ontologies we have developed contain far too few descriptive terms to make the prototype useful. Expanding the grammars and developing complete ontologies can be considered subjects for separate projects themselves, as it involves a lot of work. Especially assigning temperatures to terms; this will be a continual process.

Making domain specific ontologies available to the reasoning agent would make its range of application greater. The ability to identify how ”things“ are related in a specific field, allows the agent not only to find the temperatures of descriptions of a specific term, but also of the properties of the term in question. The lack of availability of such ontologies, especially in scarcely propagated languages like Scandinavian languages, makes us believe that the effects of the anticipated revolution of the Semantic Web is still a few years into the future.

7.5 Results and Further Work

The elements that are subject to further work are basically the shortcomings with the method and prototype. We list here the most important ones:

- **The Grammatical Rules** - The set of grammatical rules used to parse the sentences has to map better to those of the language used for parsing. Without a close-to-complete grammar, it is harder to make sense of the text.
- **Definition of Relations** - The rules used to establish relations between words has to be expanded to include the *refersToTheSameAs* in a more robust manner, and across sentences. Omitting this step, will make the agent ”miss“ a lot of descriptions of an object when it is referred to using, for instance, a personal pronoun.
- **The Ontologies** - The Descriptions Ontology and the Description Temperatures Ontology has to be expanded to include more terms. This could be done manually, semi-automatically (developing some descriptive term identifier algorithm), or automatically (having meaning-ontologies available (7.3))
- **Interconnect Domain Ontologies** - Interconnecting domain ontologies is probably the enhancement that will make the Reasoner take the biggest leap in functionality. However, the lack of these publicly available in most domains of endeavor, makes the enhancement also the most expensive one. Tailoring ontologies for a specific domain is a job for experts, and is therefore a costly process in regards of both time and money.

- **Processing** - The prototype runs on a platform proven to be inefficient regarding processing time. A reimplementation in a lower level language is recommended to be able to acquire results within reasonable times.

Adressing the listed issues, will, in our opinion, result in an agent that will be able to identify how objects are spoken of in a robust and accurate manner.

Chapter 8

Conclusion

We have developed a prototype system that uses ontologies to identify descriptive terms and their temperatures. In this process, we applied natural language processing to identify intersentence relations, and ontologies to make sense of these relations.

When we first sat down to discuss the thesis definition, natural language processing was relatively unfamiliar to us. Setting the final periods on the report, we have come to agree with the vast majority of the NLP community; natural language processing is hard. However, we feel that we have provided a solution to the problem defined in the thesis definition, in the degree possible, with the time and resources available.

In the course of the project period, we have seen that making a computer *understand* text involves a lot of work. For a computer to understand a word, it is not enough to supply a textual explanation of the word, because the explanation would not make any sense. The task is to simulate the cognitive process humans go through when we read, and formalize this into algorithms. Quite a few tools that take one or two steps in this direction are available, but we have seen that there still is a long way to go.

The proposed approach and the prototype leave several subjects open for further work. Some of these are themes for entire projects of their own. Though the prototype we have developed is working, we still think that implementing most of the proposed enhancements is required to have a really useful agent able to discover how objects are spoken of in natural language texts.

Bibliography

[1] D. J. Walker, D. E. Clements, M. Darwin, and J. W. Amtrup, "A Comparison of Paradigms for Improving MT Quality," in *Machine Translation Summit Conference*, 9 2001. [Online]. Available: http://www.eamt.org/summitVIII/papers/walker.pdf

[2] J. Reynar and A. Ratnaparkhi, "A maximum entropy approach to identifying sentence boundaries," in *Proceedings of the Fifth Conference on Applied Natural Language Processing*, 1997, pp. 16–19. [Online]. Available: citeseer.nj.nec.com/reynar97maximum.html

[3] A. Ratnaparkhi, "A Maximum Entropy Model for Part-Of-Speech Tagging," in *Conference on Empirical Methods in Natural Language Processing*, 1996. [Online]. Available: http://acl.ldc.upenn.edu/W/W96/W96-0213.pdf

[4] B. Merialdo, "Tagging English Text with a Probabilistic Model," in *Computational Linguistics, Volume 20, Number 2*, 6 1994. [Online]. Available: http://acl.ldc.upenn.edu/J/J94/J94-2001.pdf

[5] D. Jurafsky and J. H. Martin, *SPEECH and LANGUAGE PROCESSING*. Prentice Hall, 2000, vol. 1, an introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.

[6] Viterbi algorithm. [Online]. Available: http://www.math.tau.ac.il/~rshamir/algmb/00/scribe00/html/lec06/node4.h%tml

[7] B. Santorini, "Part-of-Speech Tagging Guidelines for the Penn Treebank project," *University of Pennsylvania, Dept. of Computer and Information Science*, 6 1990. [Online]. Available: ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz

[8] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Aquisition*, vol. 5, pp. 199–220, 3 1993. [Online]. Available: http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html

[9] C. technical staff, "The cyc knowledge base," *Cycorp*, August 1997. [Online]. Available: http://www.cyc.com/cyc-2-1/cover.html

[10] Laboratory for applied ontology home page. [Online]. Available: http://ontology.ip.rm.cnr.it/

[11] M. Denny, "Ontology building: A survey of editing tools," *XML.com*, 11 2002. [Online]. Available: http://www.xml.com/pub/a/2002/11/06/ontologies.html

[12] S. Holzer, *Inside XML*. New Riders Publishing, November 2000.

[13] W3c home page. [Online]. Available: http://www.w3.org/

[14] E. van der Vlist, *XML Schema*. O'Reilly, June 2002.

[15] C. Menzel, "Common Logic Standard," in *Santa Fe 2003 Metadata Forum Symposium on Ontologies*, 2003. [Online]. Available: http://cl.tamu.edu/CL-ISO.pdf

[16] *Unified Modeling Language (UML)*, Object Management Group Spec., Rev. 1.5, March 2003, formal/03-03-01. [Online]. Available: http://www.omg.org/docs/formal/03-03-01.pdf

[17] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, *OWL Web Ontology Language Semantics and Abstract Syntax*, W3C Working Draft, March 2003. [Online]. Available: http://www.w3.org/TR/owl-absyn/

[18] A. Farquhar, R. Fikes, and J. Rice, "The ontolingua server: A tool for collaborative ontology construction," *Knowledge Systems Laboratory Stanford University*, 1996. [Online]. Available: http://www.ksl.stanford.edu/software/ontolingua/

[19] R. MacGregor, "Retrospective on loom," *USC ISI*, August 1999. [Online]. Available: http://www.isi.edu/isd/LOOM/papers/macgregor/Loom_Retrospective.html

[20] *Resource Description Framework (RDF) Model and Syntax Specification*, W3C Spec., February 1999, rEC-rdf-syntax-19990222. [Online]. Available: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[21] I. Horrocks, F. van Harmelen, and P. P.-S. et al., *DARPA Agent Markup Language (DAML+OIL)*, DARPA Spec., Rev. 1.15, March 2001. [Online]. Available: http://www.daml.org/2001/03/daml+oil-index.html

[22] Wordnet home page. [Online]. Available: http://www.cogsci.princeton.edu/~wn/

[23] M. K. Smith, C. Welty, and D. McGuinness, *OWL Web Ontology Language Guide*, W3C Working Draft, March 2003. [Online]. Available: http://www.w3.org/TR/2003/WD-owl-guide-20030331/

[24] Mxpost. [Online]. Available: http://www.cis.upenn.edu/~adwait/statnlp.html

[25] The monty tagger home page. [Online]. Available: http://web.media.mit.edu/~hugo/research/montytagger.html

[26] Natural language toolkit. [Online]. Available: http://nltk.sourceforge.net/

[27] D. E. Olawsky, "The lexical semantics of comparative expressions in a multi-level semantic processor," in *Proc. of the 27th ACL*, 1989, pp. 169–176. [Online]. Available: http://acl.ldc.upenn.edu/P/P89/P89-1021.pdf

[28] D. L. McGuinness, "Conceptual modeling for distributed ontology environments," in *International Conference on Conceptual Structures*, 2000, pp. 100–112. [Online]. Available: citeseer.nj.nec.com/mcguinness00conceptual.html

[29] Grok homepage. [Online]. Available: http://grok.sourceforge.net/

[30] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: the Penn Treebank," *University of Pennsylvania*, 1992. [Online]. Available: http://www.cis.upenn.edu/~treebank/home.html

[31] University of pennsylvania. [Online]. Available: http://www.upenn.edu/

[32] Mark Watsons Homepage. [Online]. Available: http://markwatson.com

[33] Eric Brills Homepage. [Online]. Available: http://research.microsoft.com/~brill/

[34] Hidden markov models. [Online]. Available: http://www.scs.leeds.ac.uk/scs-only/teaching-materials/HiddenMarkovMode%ls/html_dev/main.html

Appendix A

Third Party Modules

A.1 Part of Speech (POS) Taggers

To decide which POS tagger to use for the implementation, two different toolkits and tree different applications have been examined. Throughout the examination, testing of these have been done to find the one the would deliver the most satisfying result, satisfying result being accurate and swift tagging. To decide which product to choose, these criterion have been used to evaluate; *Precision* and *Speed*.

There is a natural difference between the toolkits and the application. The toolkits needs to be trained to gain the same accuracy as the applications. Therefore the toolkits and the applications is described in different sections.

A.1.1 Grok

The Grok project is dedicated to developing a large collection of basic tools for use in natural language software. A particularly important aspect of Grok is that its natural language modules should follow specific guidelines, or interfaces, so that they may be freely exchanged with other modules of the same type. To accommodate this goal, Grok provides a library of modules that implement the interfaces specified by OpenNLP.

The most developed aspect of Grok is its preprocessing subsystem, which includes components for doing tokenization, sentence detection, part-of-speech tagging and name finding. The parsing system is currently being revamped to support new extensions to the Combinatory Categorical Grammar formalism which Grok uses [29].

Grok is based on XML. All rules and lexicons are described in large XML-files, which means that large lists of words would have to be put into these files to have satisfactory morphologies (see section 2.2). Listing A.1 shows the description of the word astonished.

Listing A.1: Example of a morphological description

```
<entry word="astonished" stem="astonish" pos="V" macros="@past
   "/>
<macro n="@past">
  <lf>
    <satop nomvar="E">
      <d m="tense">
        <prop n="past"/>
      </d>
    </satop>
  </lf>
</macro>
```

How the XML-files are build is scarcely documented, but there are a couple of examples that shows the use of the toolkit. Listing A.2 shows how the NNP-tag is described in Grok. It is complex and unintuitive. Still, Grok contains most of the functionality needed to reach the goals in the text-processing process.

Listing A.2: The description of the NNP-tag in Grok

```
<family pos="NNP" name="Name">
   <entry name="Primary">
      <atomcat type="np">
         <fs id="2">
            <feat attr="num">
               <featvar name="NUM"/>
            </feat>
            <feat val="+" attr="3rd"/>
            <feat attr="index">
               <lf>
                  <nomvar name="X"/>
               </lf>
            </feat>
         </fs>
         <lf>
            <satop nomvar="X">
               <prop name="[*DEFAULT*]"/>
            </satop>
         </lf>
      </atomcat>
   </entry>
   <entry name="Fronted">
      <complexcat>
         <atomcat type="s">
            <fs inheritsFrom="1">
               <feat val="fronted" attr="marking"/>
            </fs>
         </atomcat>
```

```
         <slash mode="^" dir="/"/>
         <complexcat>
            <atomcat type="s">
               <fs id="1">
                  <feat attr="num">
                     <featvar name="NUM"/>
                  </feat>
                  <feat attr="3rd">
                     <featvar name="3RD"/>
                  </feat>
                  <feat val="fin" attr="vform"/>
                  <feat val="none" attr="marking"/>
                  <feat val="-" attr="inv"/>
                  <feat val="-" attr="quant"/>
                  <feat attr="index">
                     <lf>
                        <nomvar name="E"/>
                     </lf>
                  </feat>
               </fs>
            </atomcat>
            <slash dir="/"/>
            <atomcat type="np">
               <fs id="2">
                  <feat attr="num">
                     <featvar name="NUM"/>
                  </feat>
                  <feat val="+" attr="3rd"/>
                  <feat attr="index">
                     <lf>
                        <nomvar name="X"/>
                     </lf>
                  </feat>
               </fs>
            </atomcat>
         </complexcat>
         <lf>
            <satop nomvar="X">
               <prop name="[*DEFAULT*]"/>
            </satop>
         </lf>
      </complexcat>
   </entry>
</family>
```

A.1.2 NLTK

The NL Toolkit (NLTK) is a Python package intended to simplify the task of programming natural language processing systems [26].

NLTK was designed for a course at the University of Pennsylvania. There has also been written a book, SPEECH and LANGUAGE PROCESSING [5], that uses NLTK in examples and exercises.

NLTK offers different kinds of functionality; taggers, tokenizers and parsers to mention some.

```
tagged_text_str = open('corpus.txt').read()
'John/NN saw/VB the/AT book/NN on/IN the/AT
table/NN ./END  He/NN sighed/VB ./END'

tokens = TaggedTokenizer().tokenize(tagged_text_str)
['John'/'NN'@[0w], 'saw'/'VB'@[1w], 'the'/'AT'@[2w],
'book'/'NN'@[3w], 'on'/'IN'@[4w], 'the'/'AT'@[5w],
'table'/'NN'@[6w], '.'/'END'@[7w], 'He'/'NN'@[8w],
'sighed'/'VB'@[9w], '.'/'END'@[10w]]
```

Figure A.1: Example of TaggedTokenizer

Tokenizers

NLTK has created a tokenizer called TaggedTokenizer. Figure A.1 shows how this tokenizer tokenizes sentences. If the tagger encounters a word without a tag, it assigns it the tag *none*. To train the taggers, the taggers has a method called *train*, which takes sentences tokenized by the TaggedTokenizer as input.

Taggers

To get to best result from the tagging process, NLTK offers a tagger called Backoff-Tagger. This is a tagger that is a combination of several other taggers. An example of tagger-combinations is listed bellow:

- NN_CD_tagger, which tags the text with NN if its not a number
- UnigramTagger, which is a simple statistical tagger, that tags a word with the most likely tag for that token's type.
- NthorderTagger, which is tagger that has a number *n* as argument in its constructor. In addition to considering the token's type, it also considers the POS-tags of

the n preceding tags. 0 equals UnigramTagger, 1 is called BigramTagger and 2 is called TrigramTagger.

All the taggers but the NN_CD_Tagger needs to be trained. When using them in the BackoffTagger, the order they are put into the constructor is essential for the outcome. The most accurate tagger should be first in line, and the more less accurate in their respective orders.

A.1.3 Comparing Grok and NLTK

Grok offers almost the same functionality as NLTK, but it is more complicated to put into use. The taggers in NLTK has a train-method used to train the taggers quickly, and there are a lot of pretagged texts on the Internet, suitable for the training process. This is absent from the Grok toolkit. Grok has XML-files which needs to contain all possible words in all of its possible forms to achieve analysis accuracy. No interface for creating these files is supplied, and training is a slow and tedious process.

A.1.4 The Mark Watson Tagger

Mark Watson is a consultant in Java development, artificial intelligence and natural language processing. He has written several books on Java, C++, Linux etc, and he has made a lot of products. One of these products is the POS tagger [32].

The tagger is a part of a categorizing package. It is made exclusively for English texts, and is based on a lexicon of 100.000 English words. The method used in the tagging process is a version of the TBL described in 2.4.3.

As will become apparent in the test described below, the tagger is inaccurate. The set of rules in the tagger was originally inadequate, therefore a few more were added to rid the worst errors.

A.1.5 Adwait Rathnaparkhi, MXPOST

Adwait Rathnaparkhi is working at Microsoft, and has created a tagger that he has called MXPOST. This tagger is based on the maximum entropy method described in 2.4.2. He has also created a program for sentence detection called MXTERMINATOR [24].

The MXPOST-tagger is written in Java, and is freely available in binary form (no source code). The interface towards the program is a main method, which takes an InputStream as parameter. This input could be a file, which has to contain text divided into sentences (one pr. line). The result of the tagging process is written to System.out.

The sentence boundary program divides the text into proper sentences (some preprocessing to the text is necessary to circumvent a couple of bugs in the binary).

It is possible to train the MXPOST program. MXPOST is pretrained with a lexicon of over 100000 words, and all the words is placed in files that can be edited manually, or edited by running the training module of the program.

A.1.6 The Brill Tagger

Eric Brill works at Microsoft Research, and has been a faculty member of the Department of Computer Science at U.Penn. He has developed a recognized algorithm for tagging of text [31].

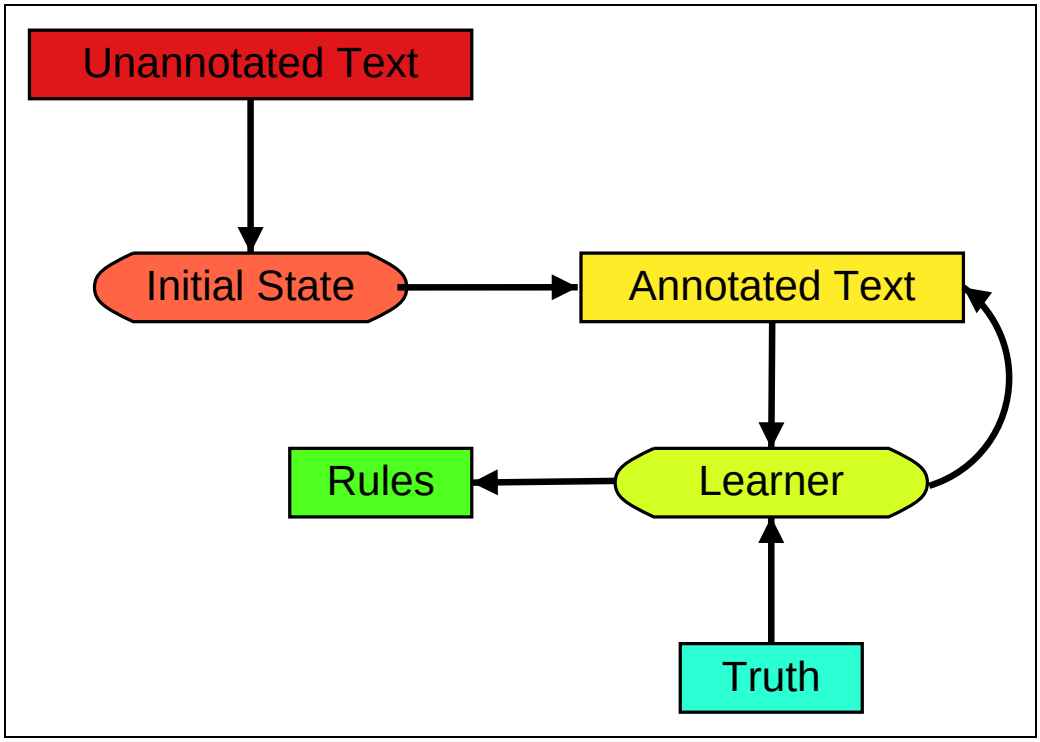


Figure A.2: Transformation-based learning

The Brill tagger is based on the TBL method in 2.4.3. It uses a transformation-based error-driven learning technique. Training of this tagger is fully automated. Figure A.2 describes the learning process.

This tagger has improved the lexicalization of unknown words. The original transformation-based tagger had relatively low accuracy at this point. The traditional tagger tags all unknown words as proper nouns if capitalized, and common noun otherwise [33].

The version of the Brill tagger we have tried is called the Monty Tagger. It is based on a lexicon of about 100000 words, and is written in Jython.

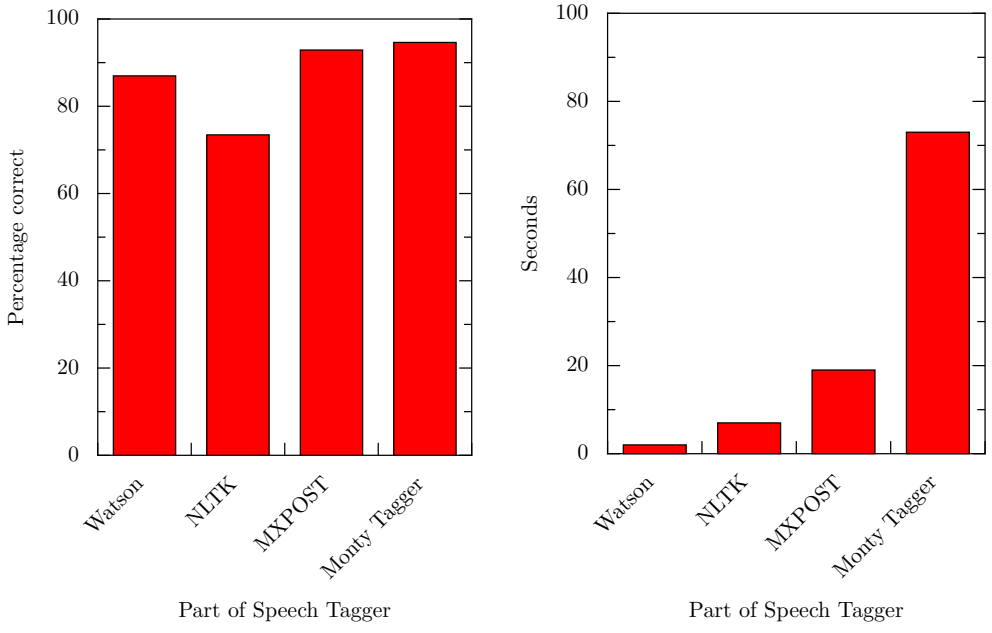
A.1.7 Comparing and Testing the Different Taggers

With the Monty Tagger, there was a test-program, that we have modified to test all the taggers we have tried. The test corpus is approximately 23.000 words, and is collected from the Penn Treebank Corpus [30].

The test was run on a Pentium III, with a 1000 MHz processor and 512 MB of RAM. The result was as follows:

TAGGER	SPEED	CORRECTNESS
Watson	2 s	Percent Agreement: 86.97%
NLTK	7 s	Percent Agreement: 73.45%
MXPOST	19 s	Percent Agreement: 92.88%
Monty	73 s	Percent Agreement: 94.63%

Table A.1: The test results



(a) The figure shows the accuracy in percent for each of the tested taggers.

(b) The figure shows the time spent for each tagger to tag 23.000 words.

Figure A.3: Bar graphs of the test results

As can be seen from the results, the Monty Tagger is the best looking at accuracy,

while the Watson tagger is extremely fast. Considering graphs A.3(a) and A.3(b), we must conclude the the MXPOST tagger gives us the most satisfying result, since it has a relatively high score, and it is fast as well. However, since accuracy of the tagger is more important than the time spent on tagging in our approach, the Monty tagger is the tagger choosen in this thesis.

A.2 The Sentence Boundary Detection Tool

A.2.1 MXTERMINATOR

As mentioned in section A.1.5, Adwait Ratnaparkhi also has developed a tool that divides text into sentences. The MXTERMINATOR is based on the maximum entropy method descibed in section 2.4.2.

Even though this method would find most of the sentence boundaries, there are a few disturbing elements in that has to be remove. These are abbreviations[1]. Abbreviations often ends with a ., which may result in confusion to the program. Therefore the MXTERMINATOR has a abbreviation list. This list is easy to edit, and to add new abbreviations.

A.3 Parsers

A.3.1 The NLTK-Parser

The parsers in NLTK has recently changed from being divided into lexicons and rules, to become a combination of both. It is called Context Free Grammars and is described in section 2.5.2

The parser is trained from a set of grammatical rules. The more precise these rules are, the fewer parse-trees is returned from the parser[2]. There is no to express all combinations of words in a natural language. A analyzing system can not include all possible rules. The solution is to use recursion. Example A.3.1 shows that a verb phrase may consist of a verb and a sentence.

Example A.3.1. VP $\rightarrow$ Verb S

[1] Abbreviations like Mr. Mrs., U.S.

[2] The parser may return zero or more parse-trees

Appendix B

Penn Treebank Tagset

These are the most important tags used in the Penn Treebank Tagset.

Table B.1: The Penn Treebank Tagset

POS TAG	DESCRIPTION	EXAMPLE
CC	coordinating conjunction	and
CD	cardinal number	1, third
DT	determiner	the
EX	existential there	*there* is
FW	foreign word	d'hoevre
IN	preposition/subordinating conjunction	in, of, like
JJ	adjective	green
JJR	adjective, comparative	greener
JJS	adjective, superlative	greenest
LS	list marker	1)
MD	modal	could, will
NN	noun, singular or mass	table
NNS	noun plural	tables
NNP	proper noun, singular	John
NNPS	proper noun, plural	Vikings
PDT	predeterminer	*both* the boys
POS	possessive ending	friend*'s*

Table B.1: (continued...)

POS TAG	**DESCRIPTION**	**EXAMPLE**
PRP	personal pronoun	I, he, it
PRP$	possessive pronoun	my, his
RB	adverb	however, usually
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	particle	give *up*
SYM	Symbol	AU (gold)
TO	to	*to* go, *to* him
UH	interjection	oh, please
VB	verb, base form	take
VBD	verb, past tense	took
VBG	verb, gerund/present participle	taking
VBN	verb, past participle	taken
VBP	verb, sing. present, non-3d	take
VBZ	verb, 3rd person sing. present	takes
WDT	wh-determiner	which
WP	wh-pronoun	who, what
WP$	possessive wh-pronoun	whose
WRB	wh-abverb	where, when

Appendix C

Implemented Ontologies

C.1 The Text Ontology OWL Implementation

Listing C.1: The OWL implementation of the Text Ontology

```
<?xml version="1.0"?>
<!DOCTYPE owl [
   <!ENTITY text "./text#" >
   <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd  "http://www.w3.org/2000/10/XMLSchema#" >
   ]>

<rdf:RDF
   xmlns       = "./text#"
   xmlns:text  = "./text#"
   xmlns:owl   = "http://www.w3.org/2002/07/owl#"
   xmlns:rdf   = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs  = "http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd   = "http://www.w3.org/2000/10/XMLSchema#">

   <owl:Ontology rdf:about="">
      <rdfs:label>Text Ontology</rdfs:label>
   </owl:Ontology>


   <owl:Class rdf:ID="Word" />
   <owl:DataTypeProperty rdf:ID="wordText">
      <rdfs:domain rdf:resource="#Word" />
      <rdfs:range rdf:resource="http://www.w3c.org/2000/10/
         XMLSchema#string" />
   </owl:DataTypeProperty>
   <owl:ObjectProperty rdf:ID="isOfClass">
      <rdfs:domain rdf:resource="#Word" />
```

```
    <rdfs:range rdf:resource="#PartOfSpeech" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="wordDescribesWord">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Word" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="wordDescribesExpression">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Expression" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="wordIsTheSameAsWord">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Word" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="wordIsTheSameAsExpression">
    <rdfs:domain rdf:resource="#Word" />
    <rdfs:range rdf:resource="#Expression" />
  </owl:ObjectProperty>


  <owl:Class rdf:ID="Sentence" />
  <owl:ObjectProperty rdf:ID="sentenceMadeUpOfWord">
    <rdfs:domain rdf:resource="#Sentence" />
    <rdfs:range rdf:resource="#Word" />
  </owl:ObjectProperty>


  <owl:Class rdf:ID="Phrase" />
  <owl:ObjectProperty rdf:ID="isOfType">
    <rdfs:domain rdf:resource="#Phrase" />
    <rdfs:range rdf:resource="#PharaseType" />
  </owl:ObjectProperty>


  <owl:Class rdf:ID="Text" />
  <owl:ObjectProperty rdf:ID="madeUpOfSentence">
    <rdfs:domain rdf:resource="#Text" />
    <rdfs:range rdf:resource="#Sentence" />
  </owl:ObjectProperty>


  <owl:Class rdf:ID="Expression" />
  <owl:ObjectProperty rdf:ID="expressionMadeUpOfWord">
    <rdfs:domain rdf:resource="#Expression" />
    <rdfs:range rdf:resource="#Word" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="expressionIsA">
```

```
    <rdfs:domain rdf:resource="#Expression" />
    <rdfs:range rdf:resource="#Phrase" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="expressionDescribesWord">
    <rdfs:domain rdf:resource="#Expression" />
    <rdfs:range rdf:resource="#Word" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="expressionDescribesExpression">
    <rdfs:domain rdf:resource="#Expression" />
    <rdfs:range rdf:resource="#Expression" />
  </owl:ObjectProperty>


  <owl:Class rdf:ID="PartOfSpeech">
    <rdfs:label xml:lang="en">Part of Speech</rdfs:label>
  </owl:Class>

  <PartOfSpeech rdf:ID="CC" />
  <PartOfSpeech rdf:ID="CD" />
  <PartOfSpeech rdf:ID="DT" />
  <PartOfSpeech rdf:ID="EX" />
  <PartOfSpeech rdf:ID="FW" />
  <PartOfSpeech rdf:ID="IN" />
  <PartOfSpeech rdf:ID="JJ" />
  <PartOfSpeech rdf:ID="JJR" />
  <PartOfSpeech rdf:ID="JJS" />
  <PartOfSpeech rdf:ID="LS" />
  <PartOfSpeech rdf:ID="MD" />
  <PartOfSpeech rdf:ID="NN" />
  <PartOfSpeech rdf:ID="NNS" />
  <PartOfSpeech rdf:ID="NNP" />
  <PartOfSpeech rdf:ID="NNPS" />
  <PartOfSpeech rdf:ID="PDT" />
  <PartOfSpeech rdf:ID="POS" />
  <PartOfSpeech rdf:ID="PRP" />
  <PartOfSpeech rdf:ID="PRP$" />
  <PartOfSpeech rdf:ID="RB" />
  <PartOfSpeech rdf:ID="RBR" />
  <PartOfSpeech rdf:ID="RBS" />
  <PartOfSpeech rdf:ID="RP" />
  <PartOfSpeech rdf:ID="SYM" />
  <PartOfSpeech rdf:ID="TO" />
  <PartOfSpeech rdf:ID="UH" />
  <PartOfSpeech rdf:ID="VB" />
  <PartOfSpeech rdf:ID="VBD" />
  <PartOfSpeech rdf:ID="VBG" />
  <PartOfSpeech rdf:ID="VBN" />
```

```
<PartOfSpeech rdf:ID="VBP" />
<PartOfSpeech rdf:ID="VBZ" />
<PartOfSpeech rdf:ID="WDT" />
<PartOfSpeech rdf:ID="WP" />
<PartOfSpeech rdf:ID="WP$" />
<PartOfSpeech rdf:ID="WRB" />

<owl:AllDifferent>
   <owl:distinctMembers rdf:parseType="Collection">
      <text:PartOfSpeech rdf:about="#CC" />
      <text:PartOfSpeech rdf:about="#CD" />
      <text:PartOfSpeech rdf:about="#DT" />
      <text:PartOfSpeech rdf:about="#EX" />
      <text:PartOfSpeech rdf:about="#FW" />
      <text:PartOfSpeech rdf:about="#IN" />
      <text:PartOfSpeech rdf:about="#JJ" />
      <text:PartOfSpeech rdf:about="#JJR" />
      <text:PartOfSpeech rdf:about="#JJS" />
      <text:PartOfSpeech rdf:about="#LS" />
      <text:PartOfSpeech rdf:about="#MD" />
      <text:PartOfSpeech rdf:about="#NN" />
      <text:PartOfSpeech rdf:about="#NNS" />
      <text:PartOfSpeech rdf:about="#NNP" />
      <text:PartOfSpeech rdf:about="#NNPS" />
      <text:PartOfSpeech rdf:about="#PDT" />
      <text:PartOfSpeech rdf:about="#POS" />
      <text:PartOfSpeech rdf:about="#PRP" />
      <text:PartOfSpeech rdf:about="#PRP$" />
      <text:PartOfSpeech rdf:about="#RB" />
      <text:PartOfSpeech rdf:about="#RBR" />
      <text:PartOfSpeech rdf:about="#RBS" />
      <text:PartOfSpeech rdf:about="#RP" />
      <text:PartOfSpeech rdf:about="#SYM" />
      <text:PartOfSpeech rdf:about="#TO" />
      <text:PartOfSpeech rdf:about="#UH" />
      <text:PartOfSpeech rdf:about="#VB" />
      <text:PartOfSpeech rdf:about="#VBD" />
      <text:PartOfSpeech rdf:about="#VBG" />
      <text:PartOfSpeech rdf:about="#VBN" />
      <text:PartOfSpeech rdf:about="#VBP" />
      <text:PartOfSpeech rdf:about="#VBZ" />
      <text:PartOfSpeech rdf:about="#WDT" />
      <text:PartOfSpeech rdf:about="#WP" />
      <text:PartOfSpeech rdf:about="#WP$" />
      <text:PartOfSpeech rdf:about="#WRB" />
   </owl:distinctMembers>
</owl:AllDifferent>
```

```
   <owl:Class rdf:ID="PhraseType" />

   <PhraseType rdf:ID="NP" />
   <PhraseType rdf:ID="VP" />
   <PhraseType rdf:ID="PP" />

   <owl:AllDifferent>
      <owl:distinctMembers rdf:parseType="Collection">
         <text:PhraseType rdf:about="#NP" />
         <text:PhraseType rdf:about="#VP" />
         <text:PhraseType rdf:about="#PP" />
      </owl:distinctMembers>
   </owl:AllDifferent>

</rdf:RDF>
```

C.2 The Descriptions Ontology OWL Implementation

Listing C.2: The OWL implementation of the Descriptions Ontology

```
<?xml version="1.0"?>
<!DOCTYPE owl [
   <!ENTITY text "./descriptions#" >
   <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd  "http://www.w3.org/2000/10/XMLSchema#" >
   ]>

<rdf:RDF
   xmlns      = "./descriptions#"
   xmlns:text = "./descriptions#"
   xmlns:owl  = "http://www.w3.org/2002/07/owl#"
   xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd  = "http://www.w3.org/2000/10/XMLSchema#">

   <owl:Ontology rdf:about="">
      <owl:imports rdf:resource="./textontology.owl" />
      <rdfs:label>Descriptions Ontology</rdfs:label>
   </owl:Ontology>


   <owl:Class rdf:ID="DescriptiveTerm" />
   <owl:DataTypeProperty rdf:ID="term">
      <rdfs:domain rdf:resource="#DescriptiveTerm" />
      <rdfs:range rdf:resource="./text#Word" />
      <rdfs:range rdf:resource="./text#Expression" />
      <rdfs:range rdf:resource="./text#Phrase" />
   </owl:DataTypeProperty>
<\rdf:RDF>
```

C.3 The Description Temperature Ontology OWL Implementation

Listing C.3: The OWL implementation of the Description Temperature Ontology

```
<?xml version="1.0"?>
<!DOCTYPE owl [
   <!ENTITY text "./descriptiontemperature#" >
   <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd  "http://www.w3.org/2000/10/XMLSchema#" >
   ]>

<rdf:RDF
   xmlns      = "./descriptiontemperature#"
   xmlns:text = "./descriptiontemperature#"
   xmlns:owl  = "http://www.w3.org/2002/07/owl#"
   xmlns:rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd  = "http://www.w3.org/2000/10/XMLSchema#">

   <owl:Ontology rdf:about="">
      <owl:imports rdf:resource="./descriptions.owl" />
      <rdfs:label>Description Temerature Ontology</rdfs:label>
   </owl:Ontology>


   <owl:Class rdf:ID="Temperature" />

   <Temperature rdf:ID="VeryCold" />
   <Temperature rdf:ID="Cold" />
   <Temperature rdf:ID="SomewhatCold" />
   <Temperature rdf:ID="Neutral" />
   <Temperature rdf:ID="SomewhatWarm" />
   <Temperature rdf:ID="Warm" />
   <Temperature rdf:ID="VeryVarm" />


   <owl:Class rdf:ID="TemperatedDescription" />
   <owl:ObjectProperty rdf:ID="temperates">
      <rdfs:domain rdf:resource="#TemperatedDescription" />
      <rdfs:range rdf:resource="#DescriptiveTerm" />
   </owl:ObjectProptery>
   <owl:ObjectProperty rdf:ID="hasTemperature">
      <rdfs:domain rdf:resource="#TemperatedDescription" />
      <rdfs:range rdf:resource="#Temperature" />
   </owl:ObjectProperty>
```

```
<\rdf:RDF>
```

Index