NTNU
Innovation and Creativity

Case-based Reasoning in Text Document Classification

Ida Sofie Gebhardt Stenerud

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Agnar Aamodt, IDI
Co-supervisor: Till Cristopher Lech, CognIt as

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The purpose of this project is to design and partially implement a system that categorizes documents with unknown content into predefined categories, based on Case-Based Reasoning (CBR). The categorization will be committed on data structures delivered by CognIT's CORPORUM system for text analysis. CORPORUM generates a semantic representation of textual documents, which consists of the key concepts and the links between these concepts. This representation is called a "light weight ontology" and is delivered in XML and/or RDF format. The CBR tool Creek will be used for categorizing the light weight ontologies delivered by CORPORUM.

Assignment given: 24. January 2007
Supervisor: Agnar Aamodt, IDI

Preface

This paper is part of a master degree at the Department of Computer and Information Science (IDI) at The Norwegian University of Science and Technology (NTNU). The project has been performed under the supervision of Agnar Aamodt at IDI NTNU and Till Christopher Lech at CognIT As.

The goal of this project is to experiment with document classification and case-based reasoning, and implement a system that demonstrates this functionality. We have used CognIT's text analysis system CORPORUM to extract information from text. This output is represented and reasoned with in Volve's knowledge-intensive case-based reasoning framework Creek.

Testing the system requires access to the Creek code, which can be found at Volve As in Trondheim. Since the code of Creek is protected, Frode Sørmo (`frode@volve.no`) or Agnar Aamodt (`agnar.aamodt@idi.ntnu.no`) must be contacted to obtain access privileges. The user will also need the executable file *CoglibTest.exe*, which is included in the .zip-file attached to this project. How to run the program is explained in the file *user_guide.txt*.

I would like to thank Frode Sørmo for patient support in practical as well as theoretical problems. I also want to thank my co-supervisor Till Christopher Lech and CognIT for providing technical assistance, an office space and vital supplies such as fruit and coffee. Finally, I would like to thank Agnar Aamodt for being a patient and helpful supervisor throughout this year.

Abstract

This work investigates document classification in Case-Based Reasoning (CBR). The investigation is exemplified by the design and implementation of a system that uses the knowledge-intensive CBR framework Creek to categorize textual cases.

The Information Extraction tool CORPORUM analyzes natural language text by extracting 'light weight ontologies' consisting of key concepts and the links between them. The output delivered by CORPORUM has been the basis of text categorization in Creek. To find the category of an unknown text case, Creek compares it to a number of already categorized texts and outputs most similar. The calculation of similarity between textual cases has been done according to Creek's existing method. The implemented program is based on a study of Textual CBR and Information Extraction, as well as an analysis of Creek's representation and reasoning functionality.

When testing the implemented system, we have observed that Creek and CORPORUM can cooperate in categorizing documents, even if their format of representing text cases is initially different. Because of differences in relation types, the general domain knowledge of Creek was not fully utilized during case matching. However, our results suggests that Creek will benefit greatly from using a text analysis tool such as CORPORUM for ontology building.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Background and Motivation

This project treats document classification, a problem where the task is to assign a document to one or more categories, based on its content. This can be done manually, semi-automatically or in a fully automatized way. Document classification is not considered a sub-field of neither Artificial Intelligence or Information Systems. It is rather a problem in information science that is best solved using techniques from several theoretical domains [Lenz 1998].

Over the past years, document classification has received increasing attention. This is much due to the exponential growth on the resources available at the world's largest document database - the Internet [Kobayashi and Takeda 2000]. However, not only Internet search engines would benefit from a more structured document collection. Enterprises typically possess large numbers of error and success reports, as well as documentation on strategies, methods and best practices. These are useful experiences that are unavailable for the end users without a proper structure.

For a document to be classified into a category, it must be decided whether or not the document belongs under the specific heading. In most cases, this is a fairly easy job for humans, but sorting an entire document collection after semantic category is an extensive job if done manually. The question is whether we can program a computer to perform the same task, and what knowledge and reasoning power would have to be available.

Case-Based Reasoning (CBR) is a method in Artificial Intelligence that solves a new problem by finding the most similar prior experience. In this project, documents will be automatically classified into predefined categories using the knowledge-intensive CBR framework Creek. Creek works by comparing structured units, and the text analysis tool CORPORUM will be used to extract structured information from natural language text.

1.2 Scientific Approach

This project is largely design-oriented, and the scientific approach varies between description/-analysis and design/implementation.

We have been assigned two Artificial Intelligence tools and a goal to make them cooperate in categorizing textual documents. Since neither of them are designed specifically for the purpose, some program code must be modified and expanded. The implementation done in this project is based on a preliminary study of the technologies involved, as well as the theoretical foundation these technologies are based upon.

The implemented system have been evaluated as a 'proof of concept', i.e a demonstration that the goal is accessible using today's technology and methods. The goal of the testing is to demonstrate the functionality of the system.

The problem of document classification is inter-diciplinary by nature and depend on techniques from different theoretical domains such as Information Retrieval, Natural Language Processing and Case-based Reasoning. The scientific basis of this project is research on textual Base-based Reasoning and Information Extraction. In addition, more distant areas such as graph theory and linguistics have contributed to the project.

1.3 Project Goals

1.3.1 Main goal

The purpose of this project is to investigate how text classification is possible using case-based reasoning. This will be exemplified by the design and partial implementation of a system that demonstrates the functionality.

More specifically, the implemented system will use the Case-based Reasoning framework Creek to categorize textual cases. The categorization will be committed on data structures delivered by CognIT's CORPORUM system for text analysis. CORPORUM analyzes text by generating a semantic representation which consists of the key concept and the links between these concepts. This representation can be delivered in XML and/or RDF format, and is called a 'light weight ontology'. Creek will be adjusted and expanded to be able to classify the light weight ontologies given by CORPORUM.

1.3.2 Subgoals

To reach the main goal, several subgoals must be reached:

Prestudy
: We will study the literature on Textual Case-Based Reasoning and Information Extraction, as well as go trough CORPORUMs text analysis functionality. In order to extend and adjust Creek, we will study its case representation and CBR process.

Design
: Based on the insights from the prestudy, we will design a system that automatically categorizes documents into predefined categories, based on Case-Based Reasoning. The system shall use CORPORUM's light weight ontologies as input and Creek as the classification tool.

Implementation
: Parts of the proposed system will be implemented as a proof-of-concept, and Creek will be adjusted and extended to handle textual cases.

Analysis
Finally, we will test and evaluate the system, before discussing the results.

1.4 System Requirements

The second subgoal in section 1.3.2 is to design a system that automatically categorizes documents into predefined categories, based on CBR. Figure 1.1 shows the general functionality that can be expected from such a system. The user feeds the system with texts and categories, and the system will output the category of the unknown text. The arrows represent the data flow between the user and the system.

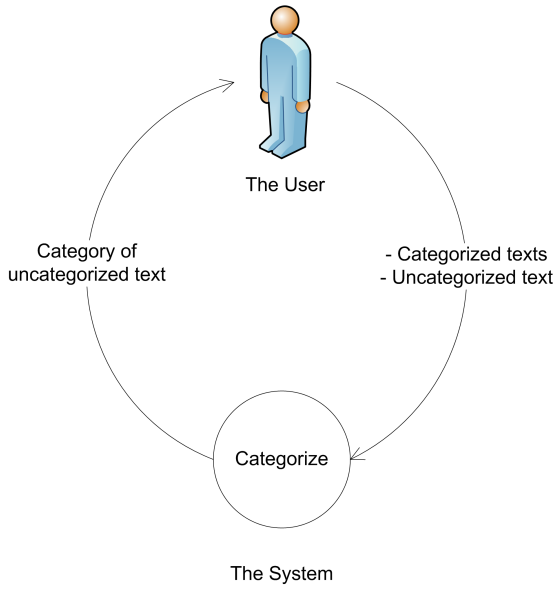

Figure 1.1: Top Level System Functionality

Automatic categorization means that the user shall provide no other input than the texts and categories, i.e the program must discover the rules of categorization by itself. However, since the task is to assign predefined categories, the system is not required to discover abstract or hidden categories. Every unsolved text will be assigned one of the categories that has been inserted by the user.

Figure 1.2 is an extended version of figure 1.1, where the system has been split up into the two applications CORPORUM and Creek. The circles represent processes, arrows are data flow between processes and components, and the broken line represent the system seen as one single unit.

The process 'Categorize' has been divided into three subprocesses, where the first, *Analyze Text*, will be performed by CORPORUM. The two other subprocesses, *Generate Cases* and *Run CBR Session*, will be implemented in Creek. In the *Analyze Text* process, the natural language text is analyzed, and structured information is extracted and stored. This must be done because texts are extremely difficult to reason with unless they are on a structured or semi-structured form [Brüninghaus and Ashley 2001]. CORPORUM delivers text in either XML or RDF format, which are both quite different from Creek's cases. *Generate Cases*

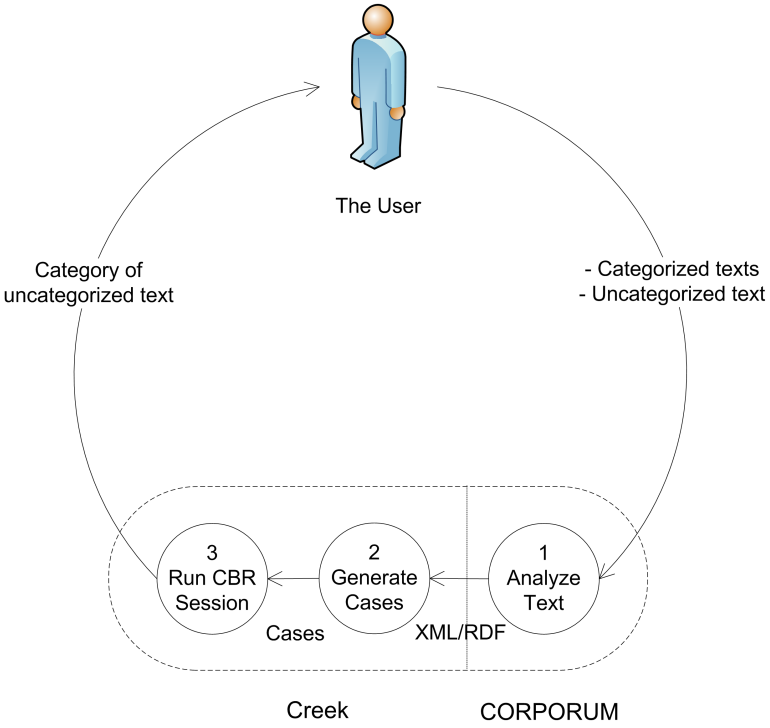

Figure 1.2: Mid-Level System Functionality

will transform the CORPORUM output to Creek cases. In process 3, *Run CBR Session*, the unsolved case is compared with all solved cases according to some similarity measure, and the output will be the most similar case.

The categorization tool should be able to recognize a document's category even if the category name is not mentioned in the text. When being confronted with input that is a duplicate of one of the experiences, this experience should obtain a match close to 100%. At CognIT's request, the system input will be short unstructured texts written in Norwegian.

1.5 Interpretation and Limits

The code that is developed during this project is only a demonstration on how Creek can classify CORPORUM's light weight ontologies. If the enterprises involved wants to either cooperate or use these results commercially, our implemented system will have to be re-implemented. When implementing, we have focused on changing and little as possible to avoid disrupting any of Creek's vital functions. Some desired system properties such as speed, performance and memory usage have not been considered very important in this project.

1.6 Structure of Thesis

Part I have introduced the thesis, while part II is a preliminary study on the theoretical domains involved in the project. We will first present Case-based Reasoning, which is the

model of categorization that will be deployed in this project. Then we will discuss Information Extraction, the process of preparing a natural language text for CBR. Then we will present ontologies and the Semantic Web, since an understanding of the concepts involved here are required for the rest of the thesis.

In part III we present the functionality of CORPORUM and Creek, and part IV presents the functional design and implementation details. The evaluation and discussion is done in part V. Finally, there is an appendix with one example of a natural language text and the corresponding RDF generated by CORPORUM.

Part II

Prestudy

Chapter 2

Textual Case-Based Reasoning

Case-Based Reasoning (CBR) refers to a model of problem solving in Artificial Intelligence which is based on the intuition that problems are often variations over already solved problems. New problems are solved by finding one or more similar past experiences, and then adapting the solutions to fit the problem. The whole CBR process is illustrated in figure 2.1.

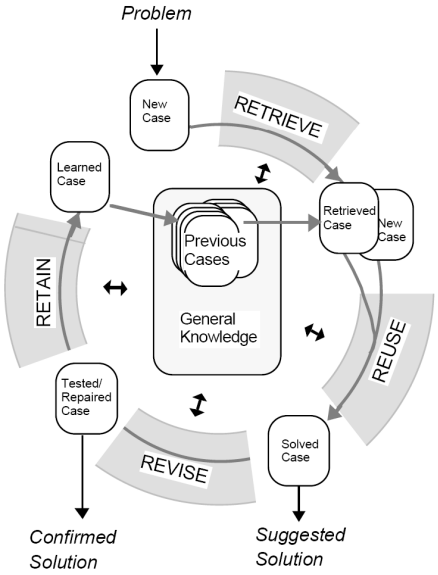

Figure 2.1: [Aamodt and Plaza 1994] The Four Steps of the CBR Cycle

A general CBR cycle can be described by the following 4 steps [Aamodt and Plaza 1994]:

1. RETRIEVE the most similar case or cases
2. REUSE the information and knowledge in that case to solve the problem
3. REVISE the proposed solution
4. RETAIN the parts of this experience likely to be useful for future problem solving

A case is a structured unit that represent some object, concept or event. It is usually described by its most salient properties and its category label(s). The most common way of representing a case is a list of properties, a so-called feature vector, where each feature can be weighted according to its relevance for the solution. In the most simple form, each feature is considered equally relevant for the case solution.

Solving a problem in CBR is based on finding the most similar experience, and this similarity between two cases can be calculated in numerous ways. When cases are represented as sets or feature vectors, a simple similarity measure between two cases is the number of overlapping features. A more sophisticated measure is the k-nearest neighbour (k-NN) algorithm, which computes the distance between the input case and all other case. The k cases with shortest distance are chosen to be the most similar ones.

In the k-NN algorithm, the distance between the cases c_i and c_j is given as [Mitchel 1997]:

$$d(c_i, c_j) \equiv \sqrt{\sum_{r=1}^{n}(f_r(c_i) - f_r(c_j))^2} \tag{2.1}$$

where n is the number of features and $f_r(c_i)$ denotes the value the rth attribute of case c_i. This method can either select the most similar case (1-NN), or it can be used to cutting of the least similar cases from further processing.

Notice in equation 2.1 that attribute r is expected to have some value $f_r(c_i)$ that can be compared with other values. When this value is numeric, the distance between two values is just their Euclidian difference. However, when the feature values are strings, there is no natural way of comparing them except from exact syntactic matching. This might lead to inaccurate results, as the same concept or event can be described with different words. In fact, more than 80% of the times, people would choose different words to describe the same concept [Deerwester et al. 1990].

2.1 Knowledge-intensive Case-based Reasoning

Knowledge-intensive CBR is an effort to reduce the problem with purely syntactic feature matches. This is done by equipping the system with general knowledge that can help select more accurate cases. In fact, the more knowledge is embedded into a CBR system, the more effective it is expected to be [Díaz-Agudo and Gonzáles-Calero 2001]. Although knowledge-intensive systems might perform better than knowledge-poor, acquiring this knowledge is a significant cost when done manually [Gabel and Stahl 2004].

Figure 2.2: [Aamodt 2004] The Knowledge-Intensiveness Dimension of CBR

CBR systems are often referred to as either knowledge-intensive or knowledge-poor. In fact, knowledge-intensiveness varies over a continuous scale, as illustrated in figure 2.2.

Knowledge can be stored within the cases, for example by attaching semantic information to the features, or in the similarity measure that is used to retrieve accurate cases. It might also be stored in a structure separate from the cases, for example an ontology or a rule base. In the knowledge container model introduced by Richter [Richter 1995], every piece of knowledge can be represented in one or more of the following categories:

- The collection of cases
- The definition of an index vocabulary for cases
- The construction of a similarity measure
- The specification of adaptation knowledge

When designing CBR application, one should reflect about which container knowledge should be added to in order obtain the maximal benefits of the program [Lenz 1998].

2.2 Textual CBR

Textual Case-Based Reasoning (TCBR) is a subfield of CBR concerned with design and implementation of CBR applications where some or all the past experiences are stored in textual format [Weber et al. 2006]. The goal is to use the textual knowledge sources in an automatic or semi-automatic way to support case-based matching.

A typical application for textual CBR is a troubleshooting or FAQ scenario. Assume that a user has a problem with some device and consults a FAQ page. Rather than having to search trough all the questions, a CBR program would utilize the fact that a similar question (though probably not equal) is likely to have been asked and solved before.

Such a troubleshooting process can be illustrated with the CBR cycle:
The user enters the problem description, and the system RETRIEVEs the problems most similar to the description. Then it identifies the differences between the problems and the description, and selects which parts should be suggested as solution. The REUSEd solution is proposed, along with a request for feedback. If feedback given by the user is negative, the solution must be REVISEd. Finally, the system RETAINs whatever turned out useful from this CBR session.

Traditionally, CBR has been focused within domains where cases have been structured, either in attribute-value pairs or complex graphs [Lenz 1998]. Natural language texts are, however, unstructured by nature. There exist many domains, such as law, medicine and diagnostics in general, where CBR techniques are particularly promising, but experiences are stored in natural language [Brüninghaus and Ashley 2005].

Today, the research on textual CBR has its main focused within four different areas [Weber et al. 2006]

1. Transforming text into structured case representation
2. Assessing similarity between textually represented cases
3. Adapting textual cases
4. Automatic generation of TCBR representations

In this project, there is no requirement that the representation shall be automatically discovered or the cases be adapted. Since the two first questions are the most important for the system we shall implement, they we will be the focus this part.

2.2.1 Transforming Text to Structured Case Representation

Transforming text to knowledge rich cases can be done in two fundamentally different ways [Wilson and Bradshaw 1999]:

1. A document can be transformed into some structure consisting of knowledge-rich features. This is done either manually or automatically with Information Extraction techniques (e.g [Lenz 1998])
2. A document containing large amounts of text can be split into several textual case components, where each component contains a part of the original text and has a defined interpretation of this text portion (e.g [Recio et al. 2005])

Representing Text

Finding a good representation for text is difficult, and no representation has so far standed out as the obvious choice for TCBR. The most basic and widely used case representation is a so-called bag of word (BOW) [Brüninghaus and Ashley 2001]. Like the name indicates, the text has been tokenized into a set of words without considering the word sequence. The features in a case represented as BOW will be either all or a selection of the words appearing in the document.

Textual cases can also be represented as feature vectors with attribute-value pairs. Attribute-value pairs are useful for representing labeled values where the labels are common for many cases. For example, a case describing a book can have features such as *Author - Amos Tversky* and *Title - Features of Similarity*.

Other approaches includes representing a text as a network structure, where features are linked to each others. This has for example been done in the law domain, representing every legal case as a labeled graphs [Cunningham et al. 2004]. The nodes correspond to important concepts, and arcs are added between concepts that are adjacent. There has also been research on representing a text in first-order language with clearly defined semantics (e.g [Gupta and Aha 2004]). However, the engineerings costs of transforming text to such a representation is beyond reach of todays technologies [Weber et al. 2006].

Finally, the case can be represented according to its document structure. When the Case-Based Reasoning framework jCOLIBRI was extended to handle textual cases, each text case was represented by an individual, which could contain other individual and thus generate a tree structure. Every individual in this approach is an array of attributes, where one attribute contains the text, and the rest of the attributes describes concepts extracted from this text. This is exactly as described in Wilson and Bradshaw's second approach. This case structure is shown in figure 2.3.

The Transformation Process

Every natural language text must be analyzed and transformed to get it on case format. This process might be as simple as dividing the text into separate words, or a complex process where

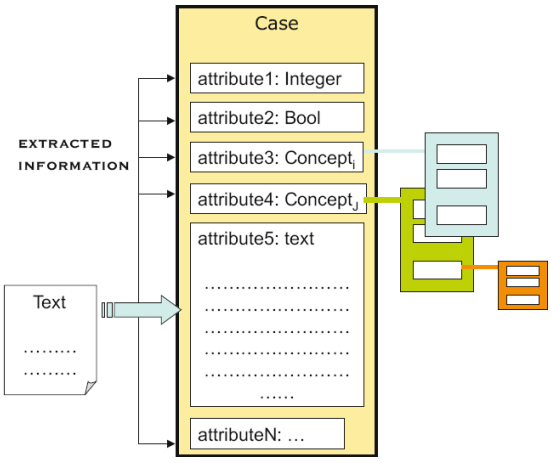

Figure 2.3: [Recio et al. 2005] jCOLIBRI Case Representation

highly knowledge-intensive representations of the textual content are generated. In general, automatic mapping between natural language text and structural case is a difficult procedure, as texts are usually created for an entirely different purpose than developing CBR applications [Davis et al. 2003]. Therefore, transforming text to a representation applicable for reasoning will usually require substantial knowledge engineering effort.

When the text has been divided into tokens, these tokens can be added directly to a case without further processing. However, this very simple approach has several drawbacks. For example, when two cases are compared, words with different grammatical tense (such as *represent* and *represented*) will not match even though they refer to the same concept. This problem can be solved by instead representing every word by its word stem. Another problem with this approach is the fact that the case will largely be populated with function words - words that appear often but carry little meaning. These can be removed either using probabilistic methods or a function word list.

M. Lenz [Lenz 1998] suggests using a layered approach that includes several modules like the above mentioned methods. Each module has its own function, and is supposed to extract one type of knowledge from the text. This approach is called Information Extraction and is one of the most generic and flexible TCBR architectures. Research suggest that this can dramatically reduce the transformation problem and make efficient and robust TCBR systems [Brüninghaus and Ashley 2001]. Information Extraction will be analyzed and described more thoroughly in the next chapter.

2.2.2 Assessing Similarity Between Textual Cases

How similarity between two cases is calculated depend largely on the representation being used. When texts are transformed to a new representation, the focus moves from comparing the texts to comparing objects of the new representation. A comparison between bag-of-word cases is usually based on set theory, where similarity is equal to the number of matching features between the two cases. There are several such formulas, where one of the most reputable is the tversky measure in equation 2.2 [Tversky 1977]:

$$S(\mathbf{A},\mathbf{B}) = \frac{f(\mathbf{A}\cap\mathbf{B})}{f(\mathbf{A}\cap\mathbf{B}) + \alpha f(\mathbf{A}-\mathbf{B}) + \beta f(\mathbf{B}-\mathbf{A})} \tag{2.2}$$

$f(\mathbf{A})$ and $f(\mathbf{B})$ are the set of all features describing the objects **A** and **B**, respectively. The constants α and β can be specified by the user or learned with machine learning algorithms.

The jCOLIBRI framework for developing textual CBR applications allow the user to choose between several similarity measures in comparing text attribute sets. They present several measures, such as [Recio et al. 2005]:

$$\text{The Jaccard Coefficient: } \frac{|o_1 \cap o_2|}{|o_1 \cup o_2|} \tag{2.3}$$

$$\text{The Overlap Coefficient: } \frac{|o_1 \cap o_2|}{min(|o_1|, |o_2|)} \tag{2.4}$$

where o_1 and o_2 are the set of attributes in text 1 and 2, respectively. $o_1 \cap o_2$ is the set of common attributes between texts 1 and 2, and $o_1 \cup o_2$ is the set of attributes that appear either in text 1, text 2 or both. $min(|o_1|, |o_2|$ is the smallest set of o_1 and o_2.

Reasoning with texts represented as networks changes the focus to calculating the similarity between two threes or graphs. Though a network based representation carry much knowledge, the transformation will probably not ease the categorization task [Lenz et al. 1998]. This is one major drawback of representing texts as networks. A simple way of comparing two labeled graphs is counting the number of matching concepts and/or relations, in a manner similar to equation 2.3. Alternatively, the algorithms may be based on graph theory, such as the greedy approach presented by Champin and Solnon [Champin and Solnon 2003].

2.2.3 Domain Knowledge in Textual CBR

Textual CBR application are usually characterized by a high degree of knowledge. In fact, the knowledge embedded in Textual CBR systems is actually what separates them from other approaches working with texts [Lenz 1998]. The knowledge container model presented in section 2.1 indicates that this knowledge can be present in several formats, such as the cases and the background knowledge.

In the process of selecting the accurate category for a text, the background knowledge might be just as important as the text itself. Consider the following text:

> Det koster lite å begrense temperaturøkningen i verden til to grader, mener FNs klimapanel. Prislappen er ca. 0,12 prosent av verdens BNP (brutto nasjonalprodukt).

For the Norwegian reader, it is clear that this text should be categorized under the heading 'klima' (eng: climate). Notice, however, that the word 'klima' is mentioned only once, and that is within the compound word 'klimapanel'. Our ability to categorize texts easily depends both on our language understanding and prior knowledge on concepts and relations in the domain.

Background and domain knowledge can be acquired and deployed in different ways, where some will be discussed in chapter 4.

2.3 Summary

This section provides the basic background of the concepts of CBR, knowledge-intensive CBR and, most important, Textual CBR.

In CBR, a problem is solved by comparing it to a number of stored experiences to find the experience that is most similar to the problem. This way of problem solving is usually used on domains where cases are structured, but can be expanded to comparing natural language texts. The goal is then to find the category that best describes an unknown text by comparing it to other categorized texts. Textual CBR works best when the text has been analyzed so that the most important information will be compared.

In the next chapter we will go trough different approaches to Information Extraction, where natural language text is automatically mined for structured information.

Chapter 3

Information Extraction

Information Extraction (IE) is a field in Artificial Intelligence where the goal is to extract structured information from unstructured text [Cowie and Lehnert 1996]. In this chapter we will present several approaches to information extraction and evaluate their ability to analyze and represent natural language, as well as their compatibility with textual case-based reasoning.

3.1 Information Retrieval

Information Retrieval (IR) deals with the representation, storage, organization of and access to information items [Baeza-Yates and Ribeiro-Neto 1999]. It is the science of searching, both for information within documents and for documents themselves. The traditional approach to information retrieval uses keyword searches and statistical techniques to retrieve relevant documents.

Note that an information retrieval system does not actually retrieve information. Rather, it searches for documents that are related to the information expressed in a query. A typical IR system ask the user to give a query with one or more words expressing the user's area of interest. The output given the user is a set of documents that the system find to be relevant.

3.1.1 Models of Information Retrieval

There are three classic IR models: Boolean, vector and probabilistic [Baeza-Yates and Ribeiro-Neto 1999].

In the most simple Boolean model, each document is represented as a set of index terms, which usually corresponds to the words contained in the document. The query is also a list of words, and the output will be documents that overlaps the query. A typical query consists of a few words, for example 'document classification case-based reasoning'. The output will then be the set of documents that contain all the words given in the query.

This model could not have been applied directly to case-based reasoning, because only texts that contains the entire query (i.e a whole text) would have been retrieved. In addition, the Boolean model has no system of ranking the documents after similarity.

A somewhat more sophisticated representation is the vector model. Each document is represented as a t-dimensional vector of index terms, where each term has a weight that indicates its importance. The query is represented in the same way.

In this model, the output will be a list of documents ranked by similarity to the query. A document can be retrieved even if it matches partially, so that the documents will not have to contain the whole query to be retrieved. It is possible to specify a threshold and retrieve only the document with a similarity degree that exceeds the threshold. The success of this model depends on how the text is transformed and which methods are used to assign relevance weights.

Both the term weights and the similarity of the vector model can be calculated in numerous ways. The weight is often based on the term's frequency and position in the text. The similarity between two texts can for example be the angle between the query and document vector, or simply the number of overlapping terms.

Inverse Document Frequency

Inverse Document Frequency (IDF) is a statistical measure that is used to describe how important a term or phrase is to the documents in a collection. The more a word appears in the general document corpus, the lower is the inverse document frequency.

The IDF formula is given as [Salton and McGill 1983]:

$$idf_i = \frac{N}{n_i} \tag{3.1}$$

where N is the number of documents, and n_i is the number of documents that contain term i.

The IDF measure is based on an intuition that words that appear in many documents are not very useful for distinguishing between relevant and non-relevant documents [Baeza-Yates and Ribeiro-Neto 1999]. IDF can be used for extracting key word that defines the content of each document, but only as long as the documents are quite different. On major drawback to IDF is that the term frequency within a single document is ignored. This is improved in the vector space model.

3.1.2 Vector Space Model

The Vector Space Model (VSM) is a specification of the vector model, where the whole document is represented as a term vector. Before transforming the text to a vector, function words should be removed from the text, either using probabilistic methods or a list over common words. Approximately 40 - 50% of the total words in a document are removed [Salton and McGill 1983].

The vector $\vec{d_j}$ for document j is defined as

$$\vec{d_j} = (w_{1,j}, w_{2,j}, w_{3,j}, \ldots, w_{n,j}) \tag{3.2}$$

The entry $w_{i,j}$ is s the weight of term i in document j. Note that i does not necessarily have to be present in j, it should rather be interpreted as how close i and j are. Alternatively, the document collection can be defined with a matrix $W = [w_{ij}]$. This matrix contains information on how important each term i is for each document j.

The weight $w_{i,j}$ is calculated as follows:

$$w_{i,j} = tf_{i,j} * idf_i \tag{3.3}$$

where $tf_{i,j}$ is the number of times term i appears in document j and idf_i is given in equation 3.1.

The vector space model is a simple and effective way to represent and compare documents. On large documents collections, the method is computationally expensive, since the indices must be re-calculated every time a new word or document is added.

3.1.3 Example of Text Classification with VSM with IDF

In 2005, there was written a master thesis on text classification with statistical IR methods [Rogstad and Ulseth 2005]. This was done using the Vector Space Model and Inverse Document Frequency. Similarly to the requirement of this project, the categories should be defined in advance, and each of the unclassified documents would be designated one of these categories.

The following categories were used: Air Traffic, Football, Handball, Movie and Music. Since a few of these domains are quite similar, some key words are likely to describe several domains. To avoid this, all documents on one domain were treated as one unit, and this unit was compared against the rest of the collection. The key words obtained was then the words that best separated the category from the other categories.

The results obtained were promising: 100% of the documents were classified correctly, and only one document fell below the classification treshold. This makes the precision rate 95%, wich must be considered very good.

3.1.4 Latent Semantic Analysis

In information retrieval, there are two fundamental problems that are unaddressed by traditional frequency-based approaches: synonymy and polysemy [Riloff and Lehnert 1994].

Two different words or phrases that expresses the same concept are called synonyms. Examples of synonyms are the words 'agreement', 'arrangement' and 'settlement'. Polysemy, on the other hand, refers to the fact that one single word can have multiple meanings. For example, the word 'deal' can refer to the a contract/agreement, a plank of pine or handing out playing cards.

Latent Semantic Analysis (LSA) it is a technique in natural language processing which automatically identifies semantic information from a corpus [Jurafsky and Martin 2000]. The LSA approach aims to solve the synonymy/polysemy problem by discovering hidden relations and concepts rather than being solely based on the words in a query.

LSA is based on the assumption that every text has an underlying built-in semantic structure. Because of the the synonymy/polysemy problem, this structure can be well-hidden and unavailable for frequency-based text analysis techniques.

The ground idea of LSA is to map each document and query vector into a lower dimensional space with associated concepts [Deerwester et al. 1990]. This process starts with a matrix W where w_{ij} is the weight of term i for document j, exactly as the as approach described in section 3.1.2. The matrix then goes trough a mathematical operation named *Singular Value Decomposition* where it is decomposed into three reduced matrices [Jurafsky and Martin 2000].

After the decomposition, the documents are represented with vectors of factor values. Accordingly, the closeness of two documents in this reduced model is determined by the overall pattern similarity rather than the exact term usage.

A result of the LSA is that abstract categories appearing in a text collection can be automatically discovered. However, this method is not yet verified to be superior in practical situations [Baeza-Yates and Ribeiro-Neto 1999].

3.1.5 Evaluation of Statistical Approaches

Statistical approaches in information retrieval have two especially desirable properties: speed and robustness against noise in the data sets. These techniques are popular because they take advantage of large document collections to identify words that are useful indexing terms automatically [Cowie and Lehnert 1996]. However, word-based techniques have some general limitations. In addition to synonymy and polysemy, there are several linguistic phenomenons that limit their success, such as [Riloff and Lehnert 1994]:

Phrases
Some phrases are combined by words that are bad indexing terms by themselves. For example, the phrase 'pass away' is synonym to dying, but neither 'pass' or 'away' are in general associated with dying.

Local Context
Some words/phrases are useful as indexing terms only in some local contexts. To retrieve texts about bank robberies, the word 'robbery' is not enough. The object of the robbery must be a bank.

Global Context
Some documents do not contain any good indexing terms. The relevance of a document may depend on the entire context of a sentence, paragraph or even whole text rather than single words. For example, the sentence 'An armed man took the money and run' refers to a robbery. However, none of the words are good indexing terms to describe robbery independently. The clear meaning of the sentence appears when the words are combined.

3.2 Natural Language Processing

It is a common assumption that natural language texts are unstructured by nature. This is not strictly true, since the words of written text seldom appear in random order. Rather, sentence composition tend to follow strict rules or patterns, which allows humans to get a deep understand the meaning of the text.

Inspired by human text understanding, Natural Language Processing (NLP) approaches usually analyze the text by dividing it into its basic components. The *principle of compositionality* states that the meaning of a natural language sentence can be derived from the meanings of it parts [Jurafsky and Martin 2000]. To illustrate, consider the following sentences:

John killed Mary

Mary killed John

It should be evident that the meaning of a sentence is based not only on the words that make it up, but also on their the ordering, grouping and inter-text relations. Instead of relying on word frequencies and other statistical measures, NLP aims to capture the text content.

3.2.1 Syntax-Driven Semantic Analysis

Syntax-driven semantic analysis is a so-called deep NLP approach where goal is to get a full understanding of the text being processed. The syntax is mapped to semantics by matching the text against language composition patterns, such as *noun-verb-noun*, which can be used to parse the sentence 'John killed Mary'. Unfortunately, most sentences have a much higher complexity level, which makes this method computationally expensive.

	Rule
1	S → NP VP
2	NP → Noun
3	NP → Proper_Noun
4	NP → Det Noun
5	VP → Verb NP

	Vocabulary
6	Noun → mammal
7	Noun → animal
8	Verb → is
9	Det → every
10	Det → an

Table 3.1: Subset of English Grammar

To analyze the more complex sentence 'Every mammal is an animal', there is needed a vocabulary that categorizes words in grammatical classes and a number of sentence composition patterns. These patterns are usually represented recursively, such as the rule base of table 3.1. The rule 'S → NP VP' means that one type of sentence S is dividable i two parts: the noun phrase and the verb phrase.

The sentence 'Every mammal is an animal' can be transformed to the parse tree in figure 3.1 using the vocabulary and grammar presented in figure 3.1. The parse tree might be used in its original form to represent text. More common, though, is transforming it to a more flexible representation such as First-Order Predicate Logic.

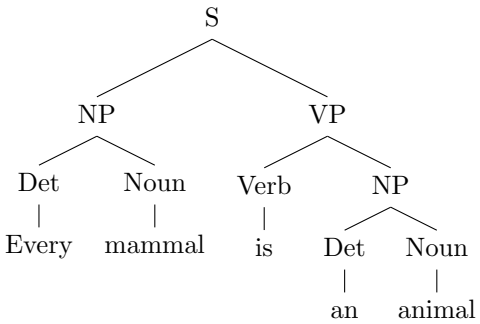

Figure 3.1: Parse Tree for Natural Language Processing

First-Order Predicate Logic

First-Order Predicate Logic (FOPC) is a popular way to represent natural language because of its flexibility, well-known semantics and sound computational basis [Nilsson 1991]. The soundness of FOPC means that what is deduced from FOPC predicates will always be true as long as the premises are true [Barwise 1977].

Classes in the domain that we are modeling can be represented in FOPC as objects, where the instances of these classes are represented as constants. Examples of objects in the above mentioned sentences are *animal* and *mammal*. The constants are John and Mary.

In FOPC, objects and constants are linked via relationships. A relationship can be expressed by a FOPC predicate, for example:

$$\text{killed (John, Mary)} \tag{3.4}$$

$$\text{killed (Mary, John)} \tag{3.5}$$

To represent the fact that either all or some instances of class has some property, we can use the $\forall$ and $\exists$ quantifiers, respectively. For example, the parse tree in figure 3.1 can be represented with the following predicate:

$$\forall x(\text{mammal}(x) \rightarrow \text{animal}(x)) \tag{3.6}$$

Predicates allows deduction of properties that are unknown but true. For example, all constants that belong to the class *mammal* will also belong to the *animal* class.

The transformation from text or parse tree to FOPC is easiest to perform by hand. It is possible to do this automatically, but this requires heavy processing capabilities [Barendregt 1981]. Syntax-driven semantic analysis in general difficult on any natural domain because of the assumption that sentences are unambiguous and free of errors [Lenz et al. 1998].

3.2.2 Shallow Natural Language Processing

In contrast to deep NLP methods such as syntax-driven semantic analysis, shallow NLP do not include full parsing of the text.

Part-Of-Speech Tagging (POS Tagging or POST) is a shallow NLP method that instead of building structured representations merely tags each word with relevant information. POST is also called grammatical tagging, and is the process of marking up the words in a text as corresponding to their grammatical group, context and relationships with other words in the text. Examples of grammatical group are noun, verbs, adjectives and prepositions.

Due to polysemy, one word might correspond to different concepts and sometimes also to different grammatical classes. The most probable grammatical tag can be found by matching ambiguous sentences to language composition patterns or by training the tagger on an already tagged document collection. The same is true for unknown words. A trained POS tagger would tag the non-existent word 'blahblahous' as an adjective, simply because the affix '-ous' is most common for adjectives [Brill 1992]. The process of part-of-speech tagging is possible to perform by hand, but is now usually computerized.

Other important shallow NLP components are tokenizer, lemmatizer, stemmer and phrase detector. The tokenizer divides a sentences into tokens, where a token is usually either a word, number or punctuation mark.

Both the lemmatizer and the stemmer returns the base form or stem of the words. A lemma is defined as the combination of the stem and its part-of-speech (POS) tag [Galvez and de Moya-Anegón 2006]. To find the lemma, the lemmatizer usually consults a list over words and their corresponding word stems. Stemming, on the other hand, is usually rule-based. Known suffices and affices are removed to obtain the word stem, without considering the context. Stemmers are easy to implement and run fast, but the accuracy is often too low for practical applications [Galvez and de Moya-Anegón 2006].

The phrase detector recognizes word phrases, which are words that naturally constitute a unit. The two most important phrase types are the verb phrase and the noun phrase. A noun phrase is a syntactic sentence unit that keeps information about a noun called the *head*. Besides the head, the noun phrase might consist of adjectives, preposition or other nouns, which all rely syntactically on the head [Kroeger 1992]. An example of a noun phrase is 'Caesar's conquest of Gaus', in which 'conquest' is the head. A verb phrase is similar to noun phrases, except that the head is always a verb.

3.2.3 Lexico-Syntactic Pattern Extraction

Lexico-Syntactic Pattern Extraction (LSPE) is a shallow rule-based approach to relation extraction. LSPE extracts relations between concepts and store these relations in a representation similar to FOPC. Similarly to Syntax-Driven Semantic Analysis, LSPE algorithms tries to match a text to several patterns. These patterns are, however, quite simple, and span over phrases rather than entire sentence.

Some patterns are easily extractable from text, such as the hyponym relation, which is a way of representing instance relationships. If A is a kind of B, then B is a *hyponym* of A, such as a *flower* being hyponym of *tulip* and *rose*.

Consider the following sentence:

```
Agar is a substance prepared from a mixture of red algae, such as Ge-
lidium, for laboratory or industrial use
```

Even though we have no acquaintance with neither Gelidium or red algae, the sentence pattern gives us a clear idea of the semantics, namely that gelidium is a type of red algae. The above pattern is also computer interpretable with the following rules [Fellbaum 1998]:

$$NP_0 \textbf{ such as } NP_1 \{, NP_2 \dots \ , \ (and \mid or) NP_i\}, \ i \geq 1$$
$$for\ all\ NP_i, \ i \geq 1, \ hyponym(NP_i, \ NP_0)$$

This allows the computer to discover the relation *hyponym*(Gelidium, red algae). Other patterns that are used to extract hyponyms from text are [Hearst 1992]:

NP $\{,\}$ **especially** $\{NP\ ,\} * \{or \mid and\}\ \ NP$

```
The colours, especially red, blue and green...
```

$\Longrightarrow hyponym(colour, \ red), \ hyponym(colour, \ blue), \ hyponym(colour, \ green)$

NP $\{,\}$ **including** $\{NP\ ,\} * \{or \mid and\}\ \ NP$

```
...some latin languages, including Spanish and French
```

$\Longrightarrow hyponym(Spanish, \ latin\ language), \ hyponym(French, \ latin\ language)$

The preferred patterns to extract relations from seem to vary greatly between applications and domains. Some programs, such as the *Snowball* framework, can learn to extract relations by scanning the text [Agichtein and Gravano 2000]. Snowball is originally domain-independent, but it is possible to add explicit rules that are useful for the specific domain being analyzed.

3.2.4 Evaluation of Rule-Based Approaches

When natural language texts are analyzed with NLP approaches, the result is a representation with a high degree of knowledge and content understanding. FOPC is the most extreme representation in terms of knowledge - when a domain model is expressed in FOPC predicates, this allows for deduction of both texts, words and concept properties.

There is, however, a trade-off between the knowledge available in the representation and the reasoning power that have to be available [Levesque and Brachman 1985]. In representing cases, parse trees are not much easier to compare than raw text [Lenz et al. 1998].

LSPE returns more shallow knowledge than deep NLP, but this method requires significantly less computation. It is easier to parse long sentences and phrases, and the methods is less prone to errors when encountering unknown words.

Because of the disadvantages related to deep NLP methods, shallow approaches are usually preferred. All the shallow NLP components in section 3.2.2 are widely used in layered text extraction approaches, as will be presented further down.

3.3 What is Structured Information?

The goal of Information Extraction is to extract structured information from unstructured text. However, this structure need not necessarily represent the semantic concepts. We will briefly represent a few other options.

3.3.1 Extracting Document Structures

A document is often naturally divided into parts such as pages, sections, chapters etc. Some parts of the document are usually more information-bearing, such as the introduction, conclusion and headers. To automatically discover this structure in a text, most researchers use methods rooted in Machine Learning and/or Information Retrieval.

One way to partition a document into structured is to track the the frequency distributions of different terms [Kulyukin and Burke 2002]. The partitions obtained will then arguably correspond to a structural organization of information in a given document. However, this approach ignores that even in a document not structured with metadata, humans tend to add signs that naturally divides text, such as headers and paragraphs.

If labeled information such as title, author and publishing year can be extracted, these are easily storable in a feature-vector, as explained in section 2.2.1. However, most of the research we have encountered (such as [Recio et al. 2005]) also analyzes the text content and attach this semantic information to their cases.

3.3.2 Extraction of Association Rules

Another structure that can be mined for is association rules. At NTNU, there is a project in progress that deals with a rule-based approach for relation extraction [Kvarv 2007]. The aim of this project is to extract a rule base from a text body, process these rules and use these to generate an ontology. In short, this is a clustering technique that determines which words tend to appear together.

The process starts with a matrix W, where w_{ij} is 1 if term i ever appears in document j, and 0 elsewhere. In other words, the weight is boolean, in contrast to the VSM approach mentioned earlier. Then follows the Apriori algorithm for association rules generation [Agrawal et al. 1993]. This algorithm aims to discover so-called *frequent item sets* from the matrix, and the association rules are then generated from these sets in an iterative fashion.

$A \Rightarrow B(s, c)$ is a valid association rule given that $\{A, B\}$ are document terms. c is the confidence and s is the support percentage [Agrawal et al. 1993]. In other words, that word A predicts B with confidence c. The further idea is to process and enhance these rules in order to use them as a basis for ontology generation.

3.4 Layered Information Extraction

In section 2.2.1, we introduced the layered approach to Information Extraction. The method starts with raw texts, and the idea is to let the text 'float' trough the different layers. Each layer has its own function and is supposed to capture one specific form of knowledge. M. Lenz suggests deploying the following layers [Lenz 1998]:

Keyword Layer
A dictionary and IR techniques such as word frequency are used to recognize important expressions in the documents. Then a part-of-speech tagger is used to obtain information about semantic word categories, and stemming information is looked up in a dictionary.

Phrase Layer
Application-specific knowledge sources, i.e relevant documents, are used to extract phrases. These knowledge sources are usually raw text and must be scanned for relevant phrases. Recognizing phrases is more difficult than just parsing keywords, so the integration of domain-experts is essential in this phase.

Thesaurus Layer
The task of the thesaurus layer is to relate different keywords to each other. Lenz suggests using a general parser such as WordNet - an ontology where nouns are organized into lexicalized concepts [Fellbaum 1998].

Glossary Layer
This layer resembles the thesaurus layer, but is mainly performed by a human domain expert. The goal of this phase is to relate application-specific terms to each other by specifying similarity relations.

Feature Value Layer
In the feature value layer, the goal is to define some attributes and their relevance to the application. These can be obtained consulting a domain experts on which attributes are normally used to describe the domain/application. Then, there is produced a similarity measure including weighting of the various features, and the different feature values.

Domain Structure Layer
The whole domain is described in this layer. With a full topic overview, one can clear out areas disjoint from the field of interest. Using the topic classification, entire sets of documents can be safely ignored and thus precision can be improved greatly.

Information Extraction Layer
In this layer, the goal is to automatically extract information in the form of attribute-value pairs. The textual components are also scanned for specific 'triggers' as an effort to generate an association rule base.

Observe how several tools and approaches are combined in this approach. A domain expert is involved in several layers, and in the Phrase Layer similar texts are scanned for key words. Both IR and shallow NLP tools such as the lemmatizer and part-of-speech tagger are used. In the final layer, relations are extracted using some type of LSPE algorithm. In addition to this, entirely external tools such as the WordNet ontology is used.

The success of this approach indicates that one single technique is not sufficient to generate structure suitable for textual CBR. Lenz analyzed the layers by subsequently eliminating higher layers and measuring the classification rates. The result was that removal of any layer would lower the performance. The biggest effect was obtained when removing the feature value layer [Lenz 1998].

This layered approach transforms each text to a structured case, in addition to adding knowledge to the domain model. Since an expert will specify both phrases and important relations between concepts, the quality of the domain model is ensured. Basically, this approach uses the best from all the approaches presented in this chapter. On the negative side, it must be mentioned that including a domain expert in the knowledge acquisition is both time consuming and expensive.

3.5 Summary

In this chapter, we have presented several approaches to analysis of natural language texts. The statistical approaches we have discussed here use only word frequencies to represent and reason with texts. This is based on an assumption that similar documents will have a bigger portion of overlapping word than dissimilar documents. This is a decent assumption, but it might be insufficient in some cases.

The rule based approached presented in this chapter aim to understand the text content and compare documents based on this content. We have also presented a layered approach that combines statistical and rule-based techniques, and gets the benefits from both the statistical and the rule-based approach.

So far, Information Extraction have only concerned transforming text to cases. However, it might be equally useful to store knowledge in a general domain model, and several of the methods here are as useful for ontology building. In the next chapter, we will examine the field of ontologies.

Chapter 4

Ontologies

An ontology refers to some kind of shared understanding with respect to a given domain. A definition of an ontology used in the AI community is 'An explicit representation of conceptualization' [Gruber 1994]. This conceptualization or world view is usually a hierarchical description of concepts and the relations that connect and describe them [Pan and Horrocks 2003].

An ontology may take a variety of forms, but will inevitably include some sort of vocabulary that defines the syntax allowed in the domain. In most cases, the ontology will also include a specification of the meaning of the words in the vocabulary. Ontologies can be expressed in any

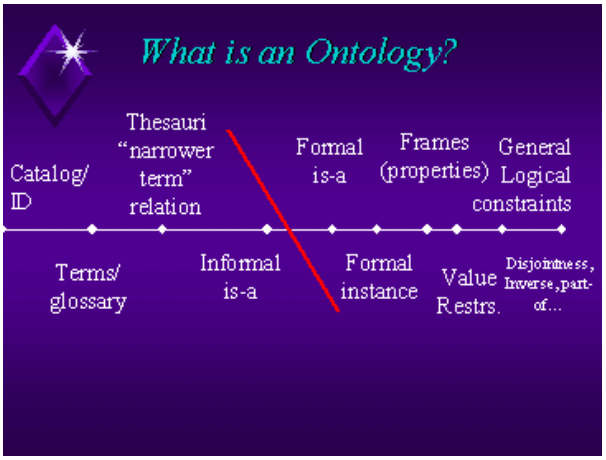

Figure 4.1: [McGuinness 2002] An Ontology Spectrum

degree of formality, from a loose informal natural language to a formal semantic specification including theorems, proof and a sound reasoning [McGuinness 2002]. Figure 4.1 illustrates the spectrum of ontologies.

The simplest form of an ontology is a finite list over words that are allowed in some language. This is called a controlled vocabulary. More sophisticated than the controlled vocabulary is a glossary, a list of words where the meaning of each word is defined in natural language.

The next point on figure 4.1 is a thesaurus. In addition to the vocabulary, the thesaurus often

includes some basic semantic relations interpretable by computer agents, such as the synonymy relationship which states that two concepts have equal meaning.

The next two types of ontologies have few surface differences. Instead, they are separated by the difference of formality of the subclass hierarchy. A subclass hierarchy is a taxonomy where the concepts are related via is-a relations. In formal taxonomies, the is-a relation have a strict use and semantic interpretation. In more informal ontologies, is-a might be interpreted as class relationships (for instance, Yorkshire Terrier is-a Dog) or instance specifications (Fido is-a Dog). It might even concern the words being used rather than physical object they refer to (Fido is-a Dog-Name).

The next point on the figure includes frames. At this point, it is often possible to separate between class nodes and object nodes, and it is possible to specify properties of both classes and objects. Even more expressive is value restrictions, where there are restrictions on what can be inserted in a property slot.

4.1 Building ontologies

There are two fundamentally different ways of designing an ontology; top-down and bottom-up. Most ontologies have some kind of top concept or relation that all other concepts are defined by. In a top-down approach, the ontology building start with defining the most general concepts, and then use the top concepts layers to define the rest of the ontology.

In the bottom-up approach, the ontology building starts with describing the details of the domain. After the relevant concepts and relations are defined, these are usually organized in some structure. The bottom-up approach is especially useful when the details are better understood than the more abstract concepts. Ontologies built bottom-up will typically reach a high level of details. Unfortunately, it is harder to control the structure and organization of concepts than in the top-down approach.

A middle-out approach combines top-down and bottom-up strategy. First, fundamental terms are manually defined, and the abstractions and details are based on this mid-level terms [Uschold and Gruninger 1996]. This approach will arguably lead to a balanced portion of details and abstraction.

Ontologies can be built manually by a domain expert, or automatically or semi-automatically by extracting concepts and relations from text. In automatic ontology building, methods from Information Extraction are frequently used. Information Extraction works best at recognizing concept as a detailed level, since high-level concepts are commonsense knowledge and therefore usually excluded from texts [Liu and Singh 2004a].

4.2 Reasoning with Ontologies

When the ontology is represented in some graph structure, the most basic reasoning methods are spreading activation and graph traversal. These methods are useful for finding out which concepts are related or computing the distance between two concepts. Alternatively, you can reason *about* ontologies, for instance by comparing two ontologies and computing the similarity between them.

More interesting, however, is reasoning that attempts to imitate how humams would utilize general domain knowledge. Reasoning *within* ontologies includes discovering inconsistencies

within the ontology as well as searching for 'hidden' concept properties. To deduce anything from the concepts expressed in an ontology, the relations must have some have some general properties that allow for drawing conclusions. Examples of such properties for relation R on the domain a, b, c are:

$$\forall a,\ aRa \tag{4.1}$$

$$\text{if } aRb \text{ then } bRa \tag{4.2}$$

$$\text{if } aRb \text{ and } bRc \text{ then } aRc \tag{4.3}$$

These are the properties of reflexivity, symmetry and transitivity, respectively. Similarity relations are often considered symmetric, so that if an object A is similar to B, then B is also similar to A. Transitive relations, such as the is-a relation, are useful for drawing sound conclusions. In general, one can say that the more transitive relations found in an ontology, the more properties are deductible [Barwise 1977].

4.2.1 Is-a and Causal Relations

Generally speaking, there are 2 types of functions that especially useful when reasoning within ontologies. Where the causal relations are mostly used for diagnostic or temporal domains, subclass or is-a relations are used in ontologies that describes relations among objects in the domain.

Reasoning With Causal Relations

Cause-and-effect analysis is an important part of commonsense reasoning. We use causal reasoning to understand every aspect of life, such as how to make dinner and why the car motor does not start.

Figure 4.2 shows a problem structure on the car starting problem. 'Normal' causal relations are represented as solid arrows. For instance, leaving the light on overnight will normally lead to a dead battery, and turning the key will normally start the car. If the battery is dead, this will normally cause the card not to start. The dashed line is blocking of one of the causal relations: a dead battery will overrun the effect of turning the car keys.

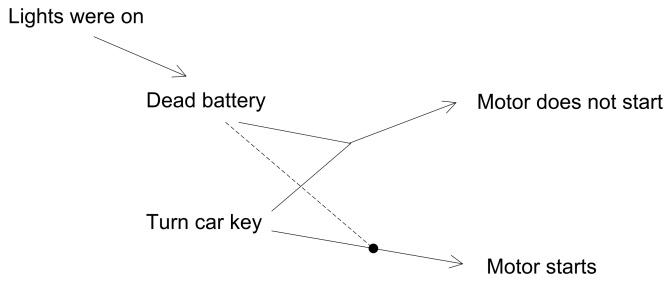

Figure 4.2: Adapted from [Konolige 1994]: A Causal Problem Structure

When causes and effects are structured as in figure 4.2, we can draw conclusions on our actions. Suppose that the lights were left on overnight, and the car keys are turned. A reasoning engine

can conclude that since the lights were on, the battery should be dead and the car will not be started by turning the key.

Reasoning With Subclass Relations

In a strict sub-class hierarchy we can deduce inheritance between objects and classes. Figure 4.3 shows such a hierarchy, where all arrows represent is-a relations. This relation is transitive, so we know that since sparrows are birds and birds are animals, sparrows are also animals.

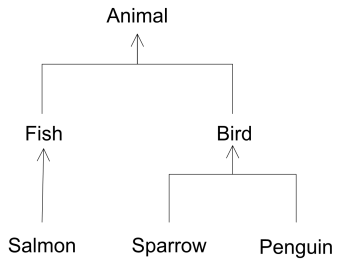

Figure 4.3: A Taxonomy with is-a Relations

The quality of whatever is deduced from an ontology depends on the degree of formality on the relations. The biggest advantages of using formal relations in ontologies are the accurate semantics and sound reasoning. However, the drawbacks are that humans are inevitably inaccurate when categorizing and comparing things, and are often willing to say that a bird is-a flying-thing even when they know that penguins are birds and can not fly. In addition, where logic is deductive, human reasoning is abductive and empirical, and conclusions are often drawn on heuristics rather than exact deduction rules.

Reasoning on causal relations is complicated by exceptions (such as the unlimited number of conditions that might cause the car not to start) and inaccurate information (for instance, keeping the lights on might cause dead batteries only 99% of the times). If the information is incomplete or insecure, causality can be represented in terms of probability of events.

When ontologies are expressed with informal relations, reasoning might lead to inaccurate conclusions. Because of the drawbacks associated with logic, some researchers suggest that formal causal relations should not be used at all, and that more practical-values relations such as similarity relations are better suited to express the information contained in an ontology [Liu and Singh 2004a].

4.3 Examples on Ontologies

We will present the three ontologies WordNet, Cyc and ConceptNet. They have been chosen because, similarly to this project, they all concern with representing and reasoning over texts and commonsense knowledge.

4.3.1 WordNet

Wordnet is a database of words, primarily nouns, verbs and adjectives. It is arguably the most popular and widely used semantic resource in the computational linguistics community [Fellbaum 1998]. The words contained in WordNet are linked by a small set of semantic relations such as the synonym, is-a and part-of relations. In other words, the concepts form a network representation, which is very flexible for representing natural language concepts. The main reasoning functionality of WordNet is word-similarity determination.

4.3.2 Cyc

The Cyc project tries to formalize commonsense knowledge into a logical framework. All knowledge is represented declaratively and described by the language CycL, which is designed especially for the purpose. Cyc can be seen as a number of small integrated ontologies called microtheories. Each microtheory captures the knowledge and reasoning power for some particular domain, such as space, time or cause-effect [Uschold and Gruninger 1996]. To ensure that the microtheories are always accurate, most of the knowledge have been manually inserted by a large number of knowledge engineers. Cyc is the most appropriate in domains where the concepts can be expressed clearly and unambiguously.

To use Cyc to reason about text, it is necessary to first map the text into the declarative CycL representation. Any ambiguity the of natural language must be resolved before representing the text in CycL. This is to avoid producing ambiguous or contradictory formulations. For this reason, Cyc is somewhat difficult to apply for practical textual reasoning tasks [Liu and Singh 2004b].

4.3.3 ConceptNet

ConceptNet has been designed as an extended combination of Cyc and WordNet. Where WordNet has its advantage in the semantic structure, Cyc is better at unambiguous logical deduction. ConceptNet have the same network structure as WordNet, but the relations are extended with more practical-valued relations such as EffectOf, PropertyOf, LocationOf, and MotivationOf. However, the scope of knowledge is general world knowledge, like the Cyc ontology. Figure 4.4 shows an excerpt of the semantic network in the ConceptNet ontology.

In contrast to both WordNet and Cyc, the knowledge of ConceptNet is generated automatically from analyzing over 700 000 sentences on the World Wide Web [Liu and Singh 2004a]. The extraction uses a pattern-approach similar to that described in section 3.2.3. For example, the knowledge extracted from the sentence 'An apple is a sweet fruit' will be both 'is-a(apple, fruit)' and 'PropertyOf(apple, sweet)'.

Several applications have been built as attachments to the ConceptNet ontology, such as a dynamic translation program that takes an input, for example the sentence 'I am at a restaurant' and automatically generates a list of concepts relevant to the situation, such as 'waiter', 'chair', 'menu' and 'eat', together with their corresponding translations.

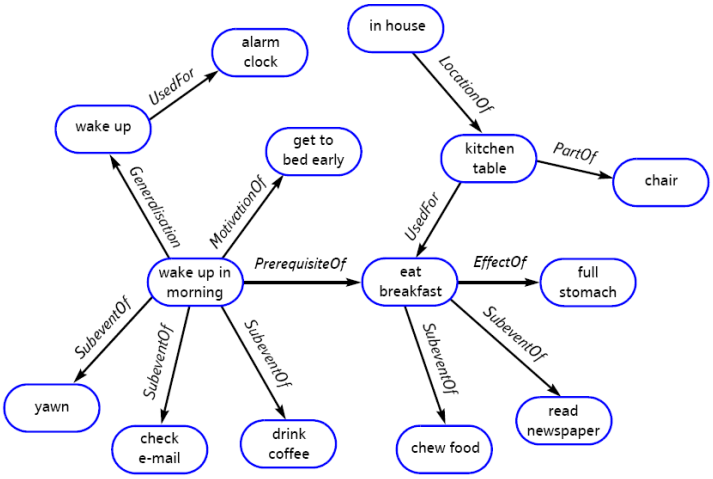

Figure 4.4: [Liu and Singh 2004b] A Subset of ConceptNet's Semantic Network

4.4 The Semantic Web

Ontologies on the web reach from taxonomies that categorize web sites to catalogues of products and services. The term 'Semantic Web' refers to a new way of structuring the web pages on the Internet. Today's web is designed primarily for the interpretation by human users. For example, to execute a particular service, such as buying airline tickets, the user must visit the different flying companies' web pages, and then search for and compare flights. In the Semantic Web, the user can instead consult one single program - the semantic web agent - and let this find the perfect ticket.

4.4.1 Metadata

To make the above mentioned scenario possible, the various web pages would have to be changed so that computer agents can interpret the semantics. The text must be structured by *metadata* - data visible to the web client, but not (necessarily) to the human user. In sum, all metadata on the Internet constitute the ontology called the Semantic Web. However, structuring documents with metadata is useful not only for web pages but for texts in general, and there is an increasing awareness of the benefits of metadata [Lider and Mosoiu 2003].

4.4.2 Resource Description Framework

The Resource Description Framework (RDF) was developed by the World Wide Web Consortium (W3C) as a standard for metadata, with the purpose of adding a formal semantic to the resources available at the Intnernet [Berners-Lee et al. 2001]. RDF provides the technology and syntax for describing how resources on the Web are connected.

The data model for RDF consist of three types of objects [Pan and Horrocks 2003]:

Resources
An RDF resource might be a Web Page, a part of a Web page or an object inaccessible

by the Internet. Resources are always named by URI's, which are kind of URL's that specify the location of the resource.

Properties
A property is a relation, attribute or characteristic that describes a resource.

Statements
Statements are triplets consisting of three parts: the subject, predicate and object. The subject is a resource, the predicate is a property and the object is either a resource or a data value.

The following lines make up an example of an RDF statement with subject 'John', object 'Mary' and predicate 'hasFriend':

```
<rdf:Description about='#John'>
        <hasFriend rdf:resource='#Mary'/>
</rdf:Description>
```

The statement relates the URIs *John* and *Mary* trough the property *hasFriend*, and the semantics of the statement is that John has Mary as a friend. An RDF document typically consist of a list of statements, which in sum make up a graph where resources are linked together by properties.

The RDF syntax is a subset of the XML language, and several technologies are defined (or planned) as an extension of RDF. How RDF relates to other languages in the semantic web can be seen in figure 4.5.

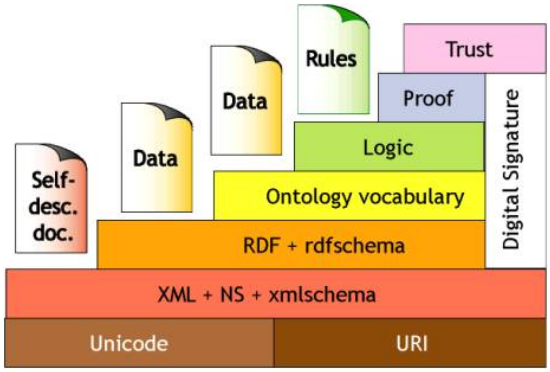

Figure 4.5: [Patel-Schneider and Fensel 2002] The Semantic Web Tower

4.4.3 RDF versus OWL

While RDF defines a simple standard for metadata, the RDF Schema (RDFS) introduces classes, is-a relationships and range restrictions [Patel-Schneider and Fensel 2002]. OWL is one of the languages that are used in the 'Ontology vocabulary' layer of the semantic web tower in figure 4.5. The intentions of OWL is to provide a richer selection of classes and properties that is suitable for expressing practical relationships between URIs.

OWL adds a broader vocabulary for describing properties and classes, like property characteristics and class relations such as cardinality and equality. Syntactically, RDF and OWL are

similar except that OWL assigns an additional meaning to certain RDF statements. This is illustrated in the following OWL code:

```
<owl:Class rdf:ID="Animal"/>
<owl:Class rdf:ID="Mammal">
        <rdfs:subClassOf rdf:resource="#Animal"/>
</owl:Class>
<Mammal rdf:ID="Weasel"/>
```

This OWL code defines the two classes 'Mammal' and 'Animal'. 'Weasel' is an instance of 'Mammal', which is subclass of the 'Animal' class. The rdfs:subClassOf is the is-a property defined in the RDFS language.

The language OWL is an extension of RDF and RDFS, so that every ontology expressed in RDF/RDFS is a valid ontology in the OWL language. However, not all information written in the OWL language will be interpretable by RDF applications.

4.4.4 Summary

During this chapter, we have presented the field of ontologies. An ontology is a structured description of a domain, and it is often formed as a labeled graph where the concepts in the domain is linked with different relations.

We have focused especially on two relations that possess the *transitive* property: the is-a and causal relation. These relations are especially useful for ontology reasoning and can be expressed in different degree of formality. The interpretation of these relations will affect whatever is deducted in the ontology.

Finally, we introduced the Semantic Web, and presented the ontology languages OWL and RDF and their applications.

Part III

Technologies

Chapter 5

Creek

Creek is a framework for knowledge-intensive CBR, developed by researchers at The Norwegian University of Science and Technology (NTNU) during the 1990s. Creek differs from most other CBR frameworks because it can utilize domain knowledge as well as a case base. This ensures that cases are matched even though they do not have the same features, as long as these features have some semantic connection.

5.1 Background and History

The original Creek framework was designed as part of Agnar Aamodt's PhD thesis in 1991 at The Norwegian University of Science and Technology. This framework has been realized in several versions, including the Java realization called TrollCreek.

Volve AS have recently bought the commercial rights to the Creek framework, with the intention of using it to predict and prevent errors in oil well drilling. Their application is called Creek and is based on the TrollCreek code and architecture. However, where TrollCreek was known for the tight coupling between cases and domain knowledge, the new Creek version has its cases stored as separate units apart from the domain model. Creek is in still under development, and changes and improvements are added continuously.

5.2 Representation and Reasoning

5.2.1 The Domain Model

The domain model of Creek is in fact an ontology - a graph structure with the entity *Thing* as its top node. Both CBR terms, domain specific concepts and relations are defined in this ontology. A selection of entities from the the top three layers of the subclass hierarchy can be seen in figure 5.1. The arrows refer to a relation type called *has subclass*. The inverse relation, *subclass of*, is similar to the is-a relation described in chapter 4.

The entities in the ontology are connected via relations. A relation is composed of a relation type (*has_color*), an origin entity (*apple*) and a target entity (e.g *red* or *green*). The relation

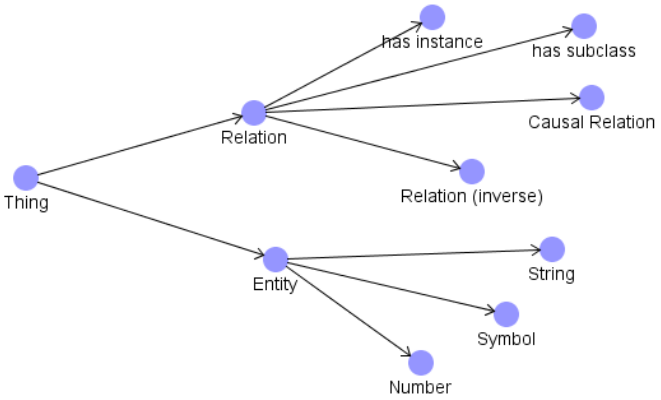

Figure 5.1: Screenshot from Creek: The General Domain Model

also has an inverse relation (*color_of*), an explanation strength (set to 0.5 per default) and some other optional fields.

5.2.2 The Case Structure

In earlier Creek versions such as TrollCreek, the entire domain was a list of relations expressed on the format the format *origin - relation type - target*. This is also true for relations concerning the cases. Cases in TrollCreek are symbolic, which means that the case must correspond to an entity in the ontology. The case entity is linked to properties via *has_finding* relations. An example of a valid relation is *Case A - has_color - blue*, given that the *has_color* relation inherits from *has_finding*.

Each case is defined by all the relations that is connected to the case entity. This very tight coupling between cases and domain knowledge is illustrated in figure 5.2.

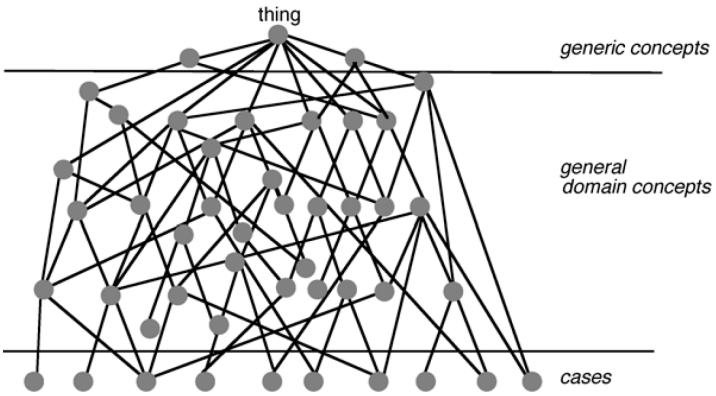

Figure 5.2: [Aamodt 2004] Integration of Cases and Domain Knowledge

It is also important to note that in older versions, all information about the cases has to be *symbolic*. This means that even numbers and string values have to be defined as entities in the ontology.

In the new Creek implementation, cases are stored separately from the general domain knowledge. The cases are now divided into symbolic organizational units called sections. Sections contains a number of features, where each feature is described by its parameter type, value and unit of measurement. This is illustrated in figure 5.3.

	Parameter Type	Value	Unit of Measurement
Person Data	Name	John	string
	Age	57	years
	Spouse	Mary	person
Work Data	Position	CFO	symbol
	Senority	15	years
	...	...	...

Figure 5.3: Creek's Case Structure

The parameter type and unit of measurement must be symbolic, but the value can be either string, numeric, URL, symbolic or another section. All the parameter types within one section must be unique, so it is not possible to have two 'Name' units within the same section.

When sections are nested, this generates a tree-structured case, where the sections correspond to those nodes in the tree that are not leaf nodes. Note that the case will constitute a Directed Acyclic Graph (DAG). A DAG is a graph whose all edges have an orientation and there are no cycles [Platt et al. 2000].

5.2.3 The CBR Process

Case matching is performed by comparing the unsolved input case to all cases in the case base and then returning the most similar. We will go trough both the old and the new case matching process.

Matching Cases in TrollCreek

In TrollCreek the procedure for comparing cases are divided into two steps, one syntactic and one semantic. First, the two cases are checked for identical features. The syntactic similarity between an input case C_{IN} and an solved case C_{RE} is the sum of the matching features' relevance. This is calculated with the following formula [Aamodt 2004]:

$$syn\_sim(C_{IN}, C_{RE}) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} syn\_sim(f_i, f_j) * rel(f_i)}{\sum_{j=1}^{m} rel(f_i)} \tag{5.1}$$

In equation 5.1, n is the number of features in C_{IN}, m is the number of features in C_{RE} and $rel(f)$ is feature f's relevance for the solution of the solved case.

In the second step, the two cases are checked for features that are related via paths in the domain model. Two features a and b are related if they are both causally related to an explanatory concept c in the domain model, so that both c *causes* a and c *causes* b - possibly via other concepts. The semantic similarity between two cases is based on the number and strength of the paths connecting them.

The final case similarity is calculated by adding the similarity obtained in the two steps, before normalizing according to the number of features. The more matching features are found between two cases, the more similar they are.

Matching Cases in Creek

When the new case structure was introduced, the case comparison procedure was changed. Now, each case goes trough a semantic transformation prior to the syntactical case matching, where the causal relations in the ontology is used to infer additional case information.

After the transformation, the input case is compared against all cases in the case base. Only sections that corresponds to the same entity in the knowledge model are compared. Within the section comparison, only entries that have the same parameter type are compared, according to the similarity measure given by the unit of measurement.

When creating a parameter type, the user can specify the similarity relation to be used. For example, numbers can be compared with Euclidian difference In today's Creek, two string and symbol values are similar only if they are equal:

$$similarity(e_1, e_2) = \begin{cases} 1 & \text{if } e_1 = e_2. \\ 0 & \text{otherwise} \end{cases}$$

Finally, the case similarity is calculated, based on the section similarities, which are in turn based on the entry similarities. The case with the highest similarity score is returned, and the results of the CBR process is illustrated in the graphical user interface.

5.3 Functionality

The two most important functions in Creek are the ontology building function and the case matching based on information found in this ontology.

When building an ontology, the user can add its own entities and relations, and integrate these with the existing domain knowledge. This can all be done directly from the graphical user interface. Entities can be opened in two different views; the frame view that shows a list of features, and a map view that shows its position in the ontology, as shown in figure 5.4. Here, the symbolic entity 0, 12 is opened in both map and frame view.

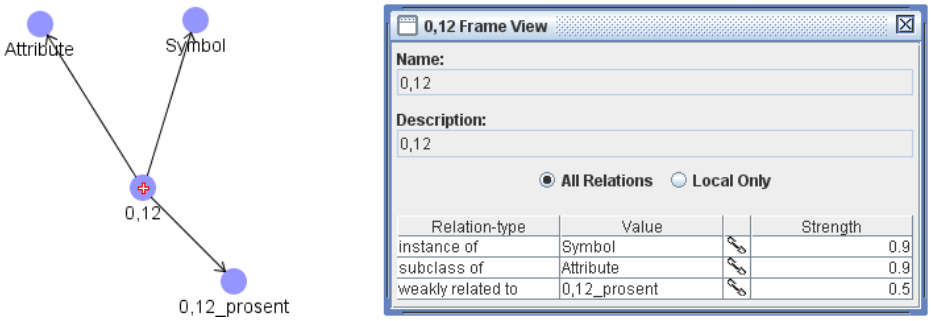

Figure 5.4: Screenshot from Creek: Map and Frame View

When working on a map view, it is possible to right-click the entity and choose which relation types to show or hide. Unfortunately, the arrows are unlabeled so that the relation name is hidden.

It is possible to save and load the entire knowledge model containing all the cases and general domain knowledge. There is also possible to export this to either OWL or JCXML (a subset of XML designed specifically for Creek). OWL is especially useful for this purpose, since it has a standard syntax and semantics. Therefore, it should, in theory, be possible to integrate the exported model with other applications or frameworks.

Similarly to the export, it is possible to import files written in OWL and JCXML. The import wizard has been written in TrollCreek code, so that the cases will be generated according to the old representation. Importing cases does not work with the new code, but since the structure of the domain model is unchanged, the import works fine as long as there are no cases.

These days, functionality is added so that cases can be imported as XML, just like an already saved knowledge model. Alternatively, the user can write the XML directly into the GUI.

Currently, there is not possible to import more than one file per program session. In other words, there is not possible to merge two imported models and generate an extended knowledge model.

5.4 Creek and Textual CBR

Right now, there is not added specific functionality for representing text cases, but, in theory, anything can be represented in Creek. The case structure is quite flexible; for example, you can add your own similarity measures. Obviously, though, the new case structure is best utilized when text is represented as some kind of list or tree structure.

If a tree structure is used to represent the cases, we must remember that the sections that make up the tree will only match if they are equal, i.e refer to the same entity. This can be useful on domains where cases are naturally divided into sections that are equal for all cases (for example; administrative data, geographical data, observable errors and tested solutions).

If a list structure is used, Creek's cases can easily be adapted to contain a bag-of-word (BOW), for example by using a flat case with only one section. Note, however, that each feature is expected to have a parameter type, which would probably be 'string' if the BOW is not processed further. The problem is then that cases are only allowed to contain one parameter type per section. If instead the token, i.e the word itself, is considered as the parameter type, a value must be attached. This value might be the frequency of the word, or its position in the text, anything that will help judge adjacency between words.

For the case to actually go trough the semantic transformation, the relations in the domain model would have to have a causal nature. As mentioned in chapter 3, causal relations are especially difficult to extract from text, and to guarantee sound computation they must be valid in all possible situations. Letting a domain expert specify the general knowledge is probably both easier and safer than extracting these using Information Extraction techniques.

Chapter 6

CORPORUM

CORPORUM is a family of products within the area of Knowledge Management, developed by CognIT a.s., where one of the key features it the ability to analyze text documents. The text analysis starts with an unstructured text written in natural language. The text is then analyzed by several modules, where most of then use techniques from Information Extraction and shallow Natural Language Processing. This resembles the layered Information Extraction approach discussed in section 3.4.

The final output of the text analysis is a so-called 'light weight ontology', which describes key concepts in the text and how these are related. In this project, we will use these ontologies in two ways. First, we will use these as basis for comparison in CBR. Second, we will merge ontologies in a general domain model.

The text analysis functionality is composite of the following modules [Lech et al. 2007]:

Tokenisation I
: In the first step, the text is broken down to paragraphs and single tokens.

Language Recognition
: For each single paragraph in the text, it is decided which language is likely to have been used. This is done by counting word frequencies and then consulting a language vocabulary.

Tokenisation II
: The paragraphs are broken down into sentences. Most sentences end with a punctuation mark followed by a capital letter, but phrases such as 'Dr. Brown' complicates the process. Therefore, several rules are deployed to set the sentence boundaries correctly. Then, short forms of phrases are extended to the original phrase (for example don't is extended to do not) and all letters are transformed to lower case. For each token, lower/upper case information and type (numeric, string, URL, proper name) are attached.

Part-of-Speech Tagging
: Part-of-speech tagging is done using a rule-based Brill tagger, which use local rules to perform the lexical analysis and word disambiguation [Voutilainen 1995]. The determination of grammatical class is based a vocabulary and statistical generalizations.

Named Entity Recognition
: In this phase, proper nouns are identified. These are either single token or multiple token grouped name collocations (such as Mary Brown), sometimes with name-indicating

tokens (such as Dr. Brown). Then an extensive grammar is consulted, to ensure that for example Robert Gordon University is tagged as organization even though Robert Gordon is tagged as male.

Key Concept Extraction and Relation Extraction
The key concepts are identified and ranked after several criteria, such as whether or not they are proper nouns or if they belong to an adjective + noun + preposition phrase. Two concepts are linked according to their rank if their distance is shorter than a constant. This constant is based on the number of sentences separating the concepts. When two concepts are related, a semantic relation is added between them. The three basic relations used today are *weakly related*, *related* and *strongly related*. CognIT have also worked on extracting other relations such as the subclass and property relation.

The idea behind this design was that the modules should be independent, so that one module could be unplugged without having any effect on the other modules' results. In reality, this have been a hard goal to reach. First of all, Tokenisation II is dependent of Language Recognition, since knowing what language is used will ease the tokenisation process. For example, the Norwegian *genetiv s* is easily confused with plural form in English.

An equally important aspect is that some modules are interdependent and difficult to run in a fixed sequence. For example, knowing the sentence boundaries are important when extracting proper nouns, but the opposite is also the case: When the proper nouns have been identified, it is easier to set the sentence boundaries correctly. Because of this, Tokenization II and Name Entity Recognition are run iteratively.

6.1 Kernel 2 and CoglibTest

The part of CORPORUM that will be deployed in this project is called Kernel 2. We originally planned to use the more recent Kernel 3, but the output is unfortunately not yet available in RDF format. Both Kernel 2 and Kernel 3 implement the module functionality, but, accordingly, the relation extraction is significantly improved in Kernel 3.

CoglibTest is a GUI (graphical user interface) built on top of Kernel 2, with most of the functionality from Kernel 2 included. A screenshot of the program is shown in picture 6.1.

CoglibTest takes an unstructured text as input, and generates the output in either XML or RDF format. Alternatively, CoglibTest can output a summary or a list of a general key words, as seen in picture 6.1. The co-reference function has not been implemented in this program version. The language options are Norwegian, English, Swedish and German.

To demonstrate CoglibTest, we have used the following text selected from the Norwegian newspaper Aftenposten: [1]

[1] http://www.aftenposten.no/nyheter/miljo/article1822401.ece

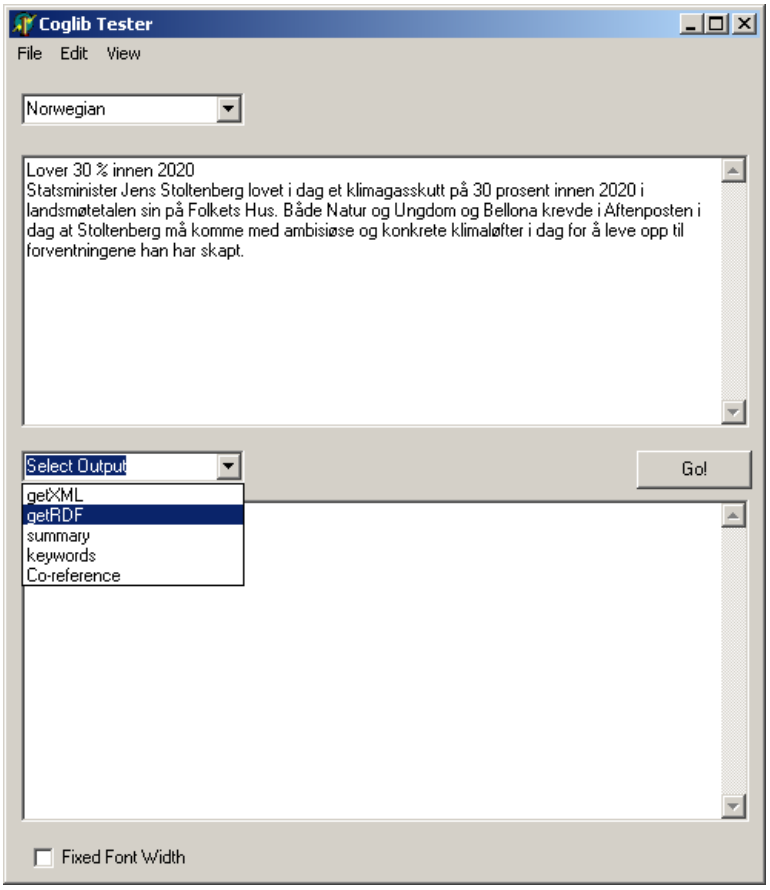

Figure 6.1: CORPORUM GUI: Screenshot from Coglibtest

Dramatisk økning
Forbrukere verden over er blitt langt mer bekymret for global oppvarming enn for bare et halvt år siden. Dobbelt så mange svarer at de er bekymret i april, som i oktober.

- Vi har ikke opplevd en så dramatisk økning i oppmerksomheten rundt klimaendringer siden slutten av 80-tallet, sier doktor Max Boykoff ved Environmental Change Institute ved universitetet i Oxford.

Han tror all oppmerksomheten rundt FNs klimarapporter, kombinert med uvanlige værmønstre slik som oversvømmelser i det sørlige Afrika og mindre snø i Alpene, har bidratt til forbrukernes sterke fokus på problemstillingen.

When generating RDF, the first line is interpreted as the title of the text. The generated RDF consists of statements in this format:

```
<rdf:Description rdf:about="april">
   <oe:relatedTo rdf:resource="#oktober"/>
</rdf:Description>
```

In addition, the output has some statements on the following format:

```
<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <do:dramatisk_økning  rdf:about="http://www.test.com#Oxford"/>
    <oe:isAbout>
</rdf:Description>
```

For files of bigger size, there is often included some of the following statements:

```
<oppvarming rdf:ID="oppvarming_001">
   <oe:hasSomeProperty>global</oe:hasSomeProperty>
</oppvarming>
```

The RDF format has a strict syntax, and because of some programming bugs in CoglibTest, the output is not well-formed[2]. Out of the three types of statement shown above, only the first type is well-formed. The other statements must be removed from the output before using the output further. The exact process is described in the attached file *user_guide.txt*.

When selecting the keyword choice, the following key words are extracted:
Alpene, Oxford, snø, universitetet, oppmerksomheten, Afrika, Dramatisk økning.

6.2 Visualizing CoglibTest Relations

We have anticipated the course of events and included some screen shots from Creek, taken right after the mentioned RDF file has been imported.

Figure 6.2 shows the graph where only the *related to* relations are highlighted. Entities that are not linked to any other entities via this relation have been excluded from the view. The arrows shows the direction of the relations. If there goes an arrow from entity A to B, this means that A is related to B, but not necessarily the opposite, since the relations do not have the symmetric property.

Similarly, figure 6.3 and 6.4 shows all the *weakly related to* and *strongly related to* relations that has been extracted from the text.

6.2.1 Observations

We will not have a full evaluation of the graphs here, only present some general observations that the reader can validate by looking at the pictures.

Figure 6.2 consists of the *related to* relations. The concepts that are extracted are all pretty sensible keyword with respect to summarizing the text. The word 'oppmerksomhet' has a central position, where almost all the other terms are related to it. This is probably because it is appearing two times and have a central placement in the text.

Figure 6.3 shows the *weakly related to* relations of the RDF file. CORPORUM have correctly identified the two proper nouns, doctor Max Boykoff and Environmental Change Institute. The phrase 'focus på problemstilling' seems to have been mistakenly identified as a proper noun or co-ocurrence phrase. CORPORUM has also extracted several adjective-substantive

[2]RDF document validation can be done on http://www.w3.org/RDF/Validator/

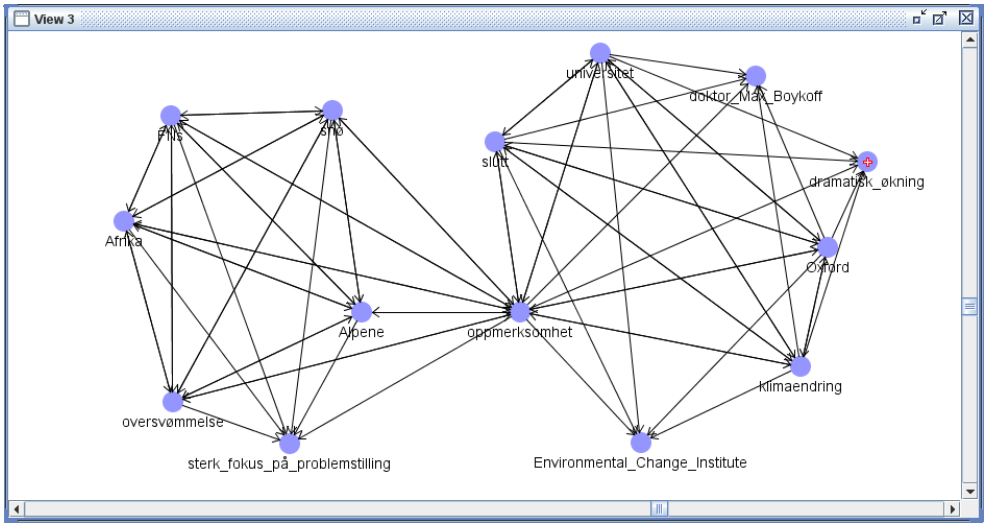

Figure 6.2: The Relation Type *Related To*

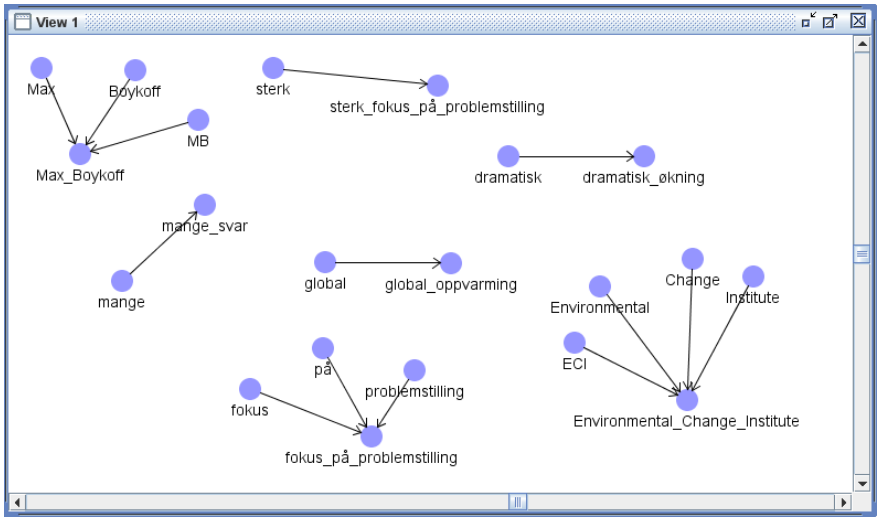

Figure 6.3: The Relation Type *Weakly Related To*

phrases (such as 'dramatisk økning' and 'global oppvarming'). For all the extracted phrases, A is weakly related to B only if B is a phrase in which A occur.

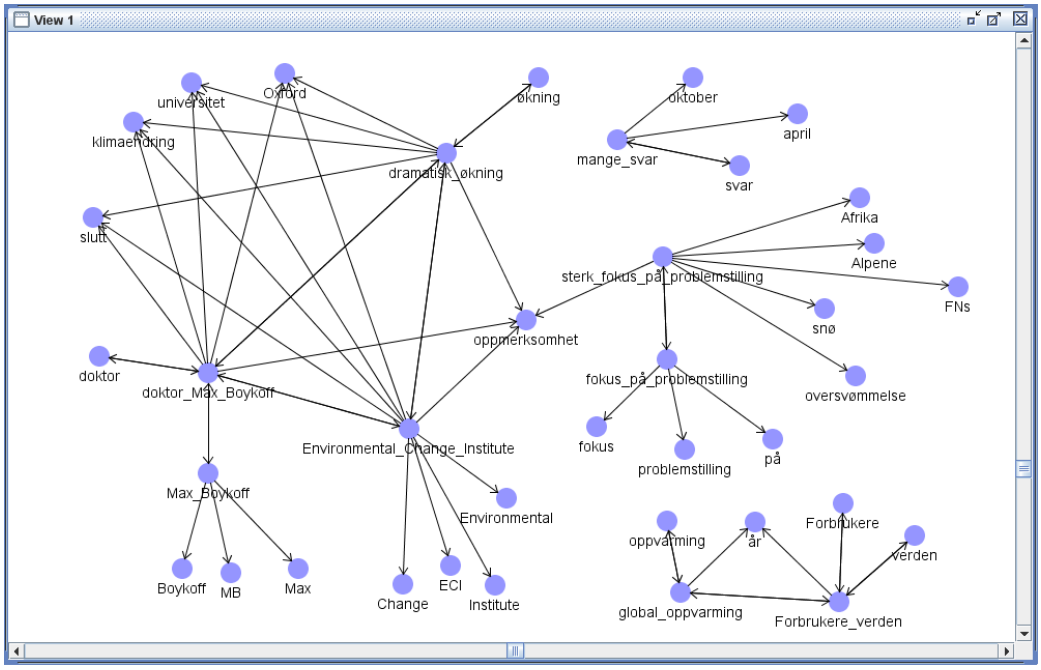

Figure 6.4: The Relation Type *Strongly Related To*

Figure 6.4 does not have a clear observable structure, but several of the concepts from the middlemost paragraph have now been included. The directions on the *weakly related to* arrows have been reversed for proper nouns, so that Max is *weakly related to* Max_Boykoff, but Max_Boykoff is *strongly related to* Max.

We think that the concepts extracted from the text are very useful. In sum, they form a good representation of the text. Most adjective-substantive phrases are recognized, except from when unknown words (such as 'værmønstre') are encountered. CORPORUM have also managed to identify the proper nouns. We are, however, less impressed by the relations that have been extracted. Very few of the relations are valid in the real world, since they are based on physical distance between the words in the text rather than real semantic links. The relations can be useful when the reasoning is restricted to graph traversal and spreading activation, but since there are no is-a or causal relations, they are not suitable for neither taxonomy building or deduction.

6.3 Other Functions in the CORPORUM Framework

Some other functions are included in the the Kernel functionality as well. First, we have the CORPORUM summarizer. This is a visualization module that is designed to present the results from the text analysis functionality. This has been implemented both as a standalone program and an integrated part of Kernel 3, but it has not been implemented in Kernel 2.

Two ontologies can also be compared. The similarity between two ontologies is calculated using an algorithm developed by CognIT. This is a graph comparison algorithm inspired by neural

networks, which uses a spreading activation technique and then counts overlapping features. The algorithm is called the *Resonance Algorithm*.

None of these extra functions have been implemented in CoglibTest, and we have therefore not been able to test these during the project.

6.4 Combining Creek and CORPORUM

In 2002, there was written a master thesis where the purpose was to make Creek and CORPORUM cooperate. This project had the main focus on improving Creek's domain model with texts analyzed by CORPORUM [Tomassen 2002].

The outcome of this project is the design and partial implementation of the Ontology Import Wizard. The wizard starts with raw texts, and trough a 6 step process the text is structured, enhanced and merged with Creek's existing domain model.

The 6 steps of the ontology import wizard are:

1. Specify Source Document
2. Identify Document Structure
3. Extract Classifications in Document
4. Extract Ontology Models
5. Enhance Concepts with Dictionaries
6. Merge with Domain Model

Step 1, 2 and 3 are manual steps where the user specifies a text and defines the text contents. In step 1, the user is supposed to specify the natural language text to use further. This can be situated on the local machine, on the Internet or inside the user's head. The files used in the project report are all medical hierarchies of symptoms and conditions.

In step 2, the structure of the document must be identified. This can be done automatically, but the user must overlook the result to see whether the automatically generated structure is correct. When done manually, the user specifies which branches of the taxonomy that is supposed to be used further.

In step 3, the user must specify the classifications of the the different concepts in the document. Symptoms and conditions must be tagged either by the ICPC or ICD-10 stadard, which are medical ontologies where each condition corresponds to an unique number. In step 4, the user selects parts of the text where relations shall be automatically extracted. All the parts are sent to CORPORUM, where they are analyzed in CORPORUM's layered architecture as described in the beginning of this chapter.

In step 5, both the extracted and the manually defined relations are enhanced using dictionaries such as WordNet and the On-line Medical Dictionary. Step 6 is the merging step, where the newly discovered knowledge is integrated with the knowledge already present in Creek's domain model.

The import function is interesting since one of our subgoals is to import CORPORUM's ontologies into Creek. Unfortunately, step 5 and 6 of the cycle was never implemented, so we can not use the results directly. However, some important observations were made when analyzing CORPORUM's ability to extract relations from text.

The big question is of course whether CORPORUM can extract relations that are valid in the *real world*, or only within the *document world*. To check this, the following medical text were used:

> Chest pains are pains of all types experienced as originating from the chest. They can occur in a large number of conditions. The dominating cause of chest pains is muscle tension in the chest wall. Such pains are closely related to anxiety, depression or stress of a psychosocial nature.

This text was made into an ideal model that expressed the correct causal and subclass relationships between the key concepts. After creating the ideal model, this was compared to a model extracted by CORPORUM. It must be mentioned that the ideal model contained relation types that CORPORUM is unable to extract, such as causal relations. Nevertheless, one should expect that the extracted model would grasp some of the text meaning.

The obtained results were summarized as follows [Tomassen 2002]:

- Most of the key concepts were included in CORPORUMS extracted model. However, the most central concept, *pain*, was actually left out.
- The two models had few relations in common.
- There were no consistent method of mapping the automatically generated model to the ideal model.

The conclusion is that even though the extracted model had grasped some of the essence of the text, addition information was required to perform an adequate knowledge acquisition process. The extracted model gives an indication of how concepts can be related, but this does not necessarily apply to how the concepts are related in the real world.

Part IV

Results

Chapter 7

Functional Design

This chapter presents the functional design of the system we are about to implement. This chapter is based on the prestudym, the functional requirements in section 1.4 and the analysis of Creek and CORPORUM in chapter III.

7.1 Main System Components

The main components in this system are the knowledge-intensive CBR framework Creek and the text analysis tool CORPORUM, which is implemented in the executable program CoglibTest. CoglibTest delivers the light weight ontologies in two different formats; XML and RDF. We have chosen to use RDF output in this project. This is because an OWL import wizard is already implemented in TrollCreek, which have made it possible for us to reuse some of the import functionality.

Having a direct communication between Creek and CORPORUM turned out difficult since the tools we have been assigned are not designed for communicating. CoglibTest is locked for modification, and we were unable to make Creek communicate directly with the executable file. For this reason, we have chosen to let all communication between CORPORUM and Creek go trough the user. The RDF delivered by CoglibTest must be temporarily stored before manually importing it into Creek. See figure 7.1 for an illustration of the responsibility distribution between the user and the different system components.

7.2 Text Case Representation

Because CORPORUM delivers its light weight ontologies in graph format, we have chosen to use cases structured as networks. Each case consist of two parts; the RDF graph and the case status, which is either 'Solved' or 'Unsolved'. Solved cases have one more entry, namely the solution of the case, which is always the category of the corresponding text. This category is found in the first line of the text when it is inserted in CoglibTest.

The far most common for Textual CBR is having a one-to-one cardinality between texts and cases, so that each text is represented in one case. In this project, however, we have experimented with two different solutions:

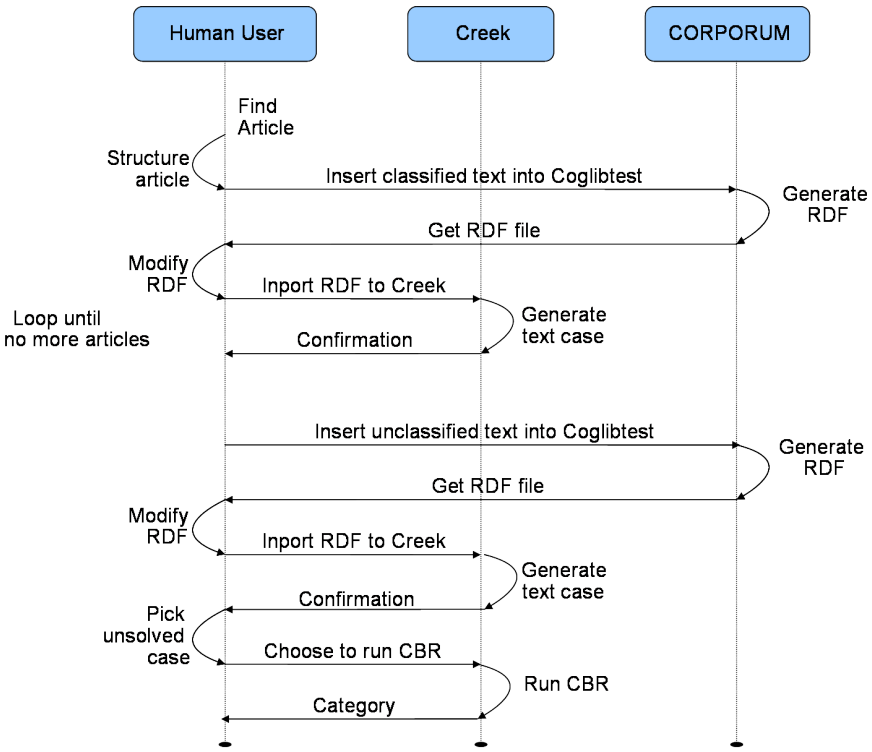

Figure 7.1: Sequence Diagram

1. Each case represents one single text
2. Each case represents one entire category, which possibly consists of several texts

For simplicity, we will refer to the solutions as solution 1 and solution 2. The difference between solution 1 and 2 is illustrated in figure 7.2. The first row of figure 7.2 shows three texts, where two of these have category A and one has category B. In step 2, the texts have been transformed to RDF graphs with nodes (concepts) and edges (relations between the concept). In step 3, the cases have been generated, consisting of two parts: the RDF graph and the category label. This corresponds to the standard CBR approach in solution 1. In the fourth row, the cases have been generated according to solution 2. The two RDF files that are classified under the same category are combined to generate a case that represents the whole category, rather than just one single text.

7.3 RDF Import and Merging

It is possible to import any RDF file to Creek, as long as the file contains only well-formed statements. We have implemented the RDF import function, and at the same time made it possible to import more than one RDF file into the knowledge model. This means that the concepts and relations of the imported RDF file will be merged with knowledge from other cases. Regardless of which of the two solutions have been chosen, all concepts and relations are added to the ontology as well as to the case.

The RDF merging has some interesting consequences for the ontology. For instance, assume

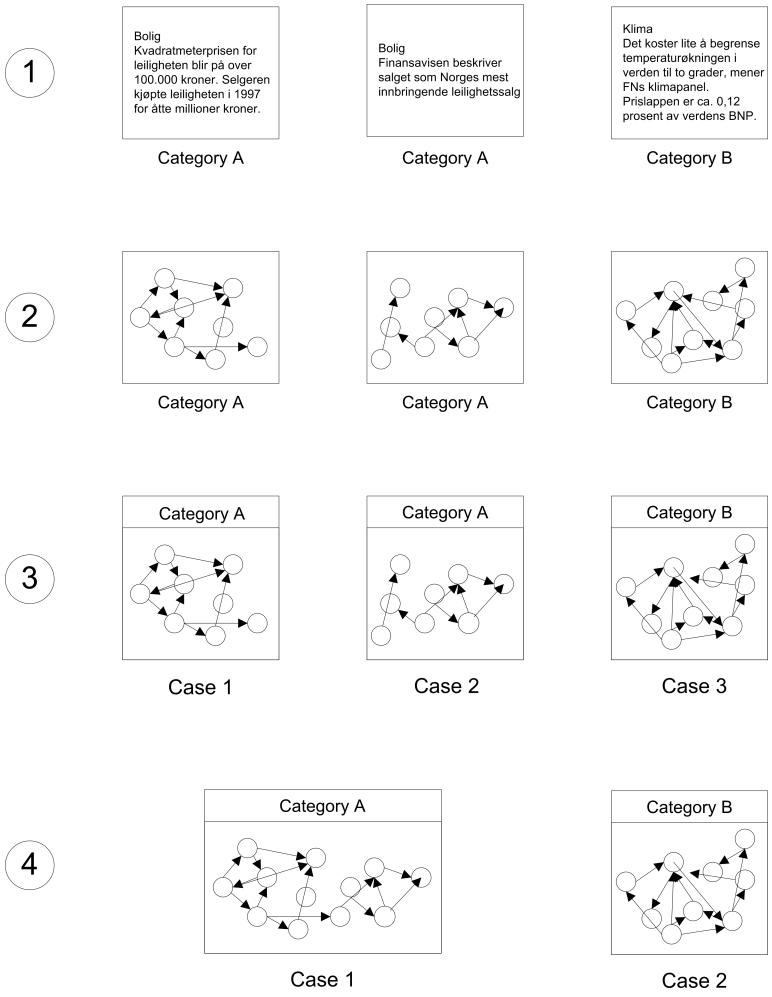

Figure 7.2: The Two Solution of Case Representation

that some concept is described in two different texts. The corresponding RDF files will probably describe this concept with different relations. When the files are merged, this will result in an extended representation where common concepts are described with relations extracted from more than one text.

7.4 The CBR Matching

Comparing cases in Creek is done according to Creeks built-in CBR matching, as explained in chapter 5. In both solution 1 and solution 2, the input case and the solved cases have the form of labeled directed graphs. When finding the most similar case, the focus has now been moved to finding the most similar graph.

The solution on calculating the similarity between two network based cases has been inspired by Frode Sørmo's work on Concept Maps. The definition of a Concept Map is an ordered set

$M = \{V, L, A\}$ with the following properties [Sørmo 2007]:

$$V \text{ is a definite set of vertices)} \tag{7.1}$$
$$L \text{ is a definite set of link labels} \tag{7.2}$$
$$A \subseteq V \times L \times V \text{ defines the labeled arcs} \tag{7.3}$$

Translated to this domain, the relations are the arks A, the concepts are the vertices V and the relation types (*related to*, *weakly related to* and *strongly related to*) are the link labels L.

Frode Sørmo have defined the similarity between two concept maps $M_1 = (V_1, L_1, A_1)$ and $M_2 = (V_2, L_2, A_2)$ as [Sørmo 2007]:

$$sim(M_1, M_2) = \frac{|A_1 \cap A_2|}{|A_1 \cup A_2|} \tag{7.4}$$

This is in fact similar to the Jaccard Coeffisient presented in section 2.2.1. We have used this similarity measure in case comparison, but it has been modified because the original measure only considers the arcs A - i.e the relations. When evaluating CORPORUM's graphs we found the extracted entities are usually more in touch with the real content than the extracted relationships. Therefore, we wanted to include both the relations A and the concepts V. This will ensure that two graphs are matched if all relations differ, as long as they have some entities in common.

The modified Jaccard Coeffisient used in this project is given by the following equation

$$sim(M_1, M_2, c) = c\frac{|A_1 \cap A_2|}{|A_1 \cup A_2|} + (1 - c)\frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \tag{7.5}$$

where A is the relations, V is the entities and $c \in [0, 1]$. We have set c to $2/3$, meaning that the number of common entities yield one third of the total similarity. This is not really scientifically justified, c could have been any number between 0 and 1.

As described in section 5.2.3, there is a CBR step that tries to deduce hidden knowledge by looking for causal relations within the ontology. In this project, the only relations that appear in the ontology are *related to*, *weakly related to* and *strongly related to*, and none of these are causal. Therefore, no new properties will ever be deduced here. See chapter 10 for a broader discussion on this point.

7.5 Process Decomposition

Figure 1.2 in the Requirement Specification indicates that the only time the user directly communicates with the system is when inserting the input and receiving the output. This is a simplified presentation, since the user will necessarily have to give input to the GUI several times, for example when initiating a CBR session. In addition, the user must manually transfer the RDF files from CoglibTest to Creek. This is all illustrated in figure 7.3.

All arrows in figure 7.3 refers to data flows between processes. As earlier, the user inserts categorized and uncategorized texts, but the output delivered in RDF must be manually modified and stored before it is imported into Creek. The 'Solution' arrow is a signal from the delivered by the GUI, where the user is asked to choose which of the two solutions to use. When the

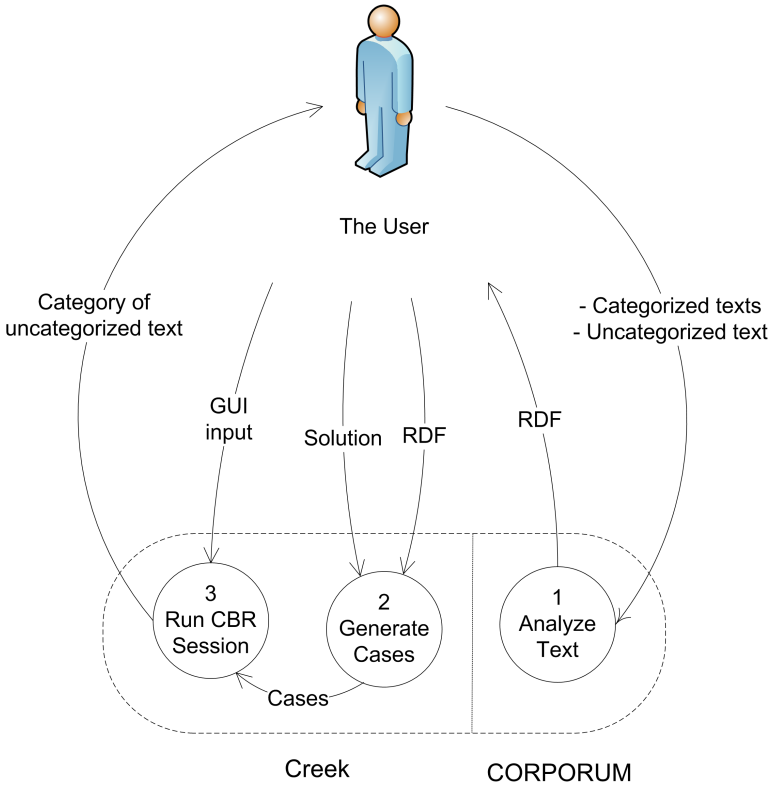

Figure 7.3: Mid-Level System Functionality with Manual Steps

user has imported all the cases, he or she can double-click an unsolved case to start a case comparison. This is the 'GUI input' arrow. The category is delivered to the user from the graphical user interface.

The three processes in figure 7.3 have been analyzed and decomposed in figure 7.4, 7.5 and 7.8. All circles refer to processes that transform the input to some output, usually in another format. The data flowing in and out of the processes have been illustrated with directed arrows.

7.5.1 Process 1: Analyze Text

Figure 7.4 shows the process of analyzing the text, decomposed into 7 subprocesses. The 6 first subprocesses correspond to the layers described in chapter 6. Since process 1.3 and 1.5 are run iteratively, this presentation does not describe the exact order, but is still useful for demonstrating the data flow between layers. The seventh circle is the process of storing the relations as RDF statements on the format described in section 4.4.2.

7.5.2 Process 2: Generate Cases

Figure 7.5 is a decomposition of the case generation process. The input is both the RDF file delivered by process 1, and a variable expressing which solution is being used. This variable is

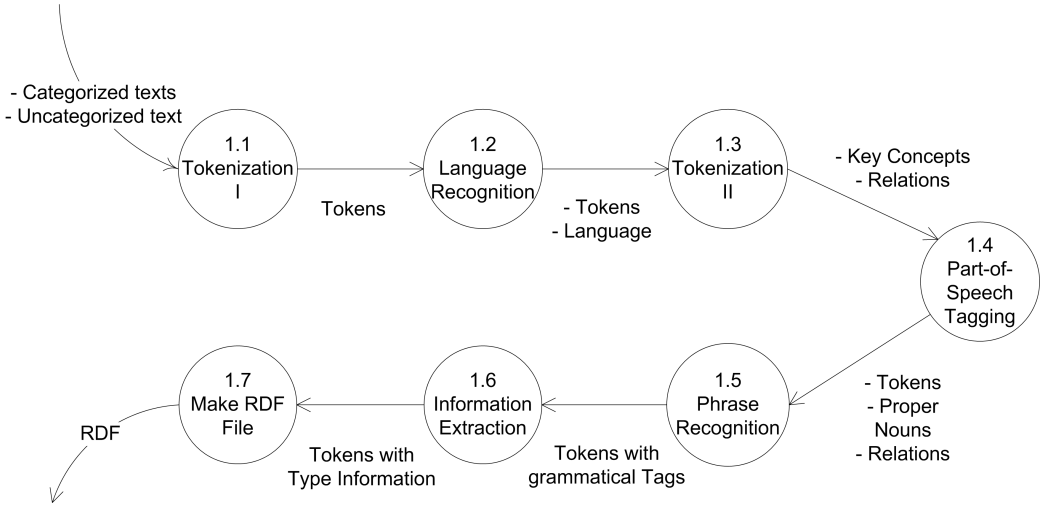

Figure 7.4: Decomposition of Process 1: Analyze Text

specified when the user imports the first RDF file and the GUI asks the user to choose between solution 1 and 2.

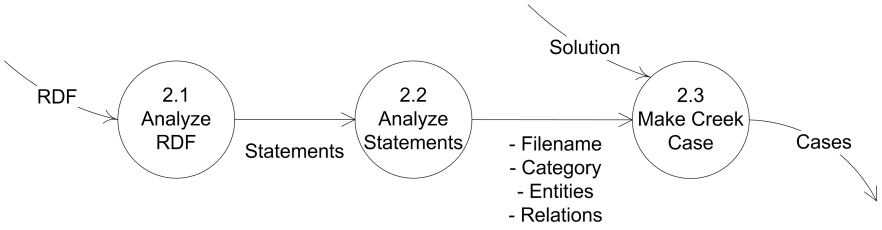

Figure 7.5: Decomposition of Process 2: Generate Cases

Process 2.1, *Analyze RDF*, and process 2.2, *Analyze Statements*, are described in pseudocode together in figure 7.6. Process 2.3, *Make Creek Case*, is described in pseudocode figure 7.7.

Basically, *Analyze RDF* reads an RDF file and outputs all the well-formed RDF statements in the file, and *Analyze Statements* divides the statements into subject, predicate and object. The title and file name of the RDF file is extracted as well. All subjects and objects are added as entities in the ontology, and the subject-predicate-object triplets are added as relations. In addition, all relations and entities are listed and outputted to the next process.

Process 2.3, *Make Creek Case*, takes the file name, category, entity list and relation list from process 2.2 as input, and outputs a generated case, as illustrated in the pseudocode in figure 7.7. This pseudocode says that if the user has chosen solution 2, the whole category will be represented in a case. Therefore, the system searches for a case with the same category as the input case, and if this exists, the entities and relations will be added to this case. In all other cases, a new case will be generated, and entities and relations are added here.

```
input = RDF file F

for all statements ST in F
    analyze ST
    string S = the subject of ST
    string P = the predicate of ST
    string O = the object of ST

    if P equals 'Title'
    then
        string FileName = S
        string CaseCategory = O
        end
    end
    else
        entity E1 = makeEntity(S)
        entity E2 = makeEntity(O)
        relation R = makeRelation(S, P, O)
        add E1 and E2 to EntityList
        add R to RelationList
end

output = FileName, CaseCategory, EntityList, RelationList
```

Figure 7.6: Pseudocode for Process 2.1 and 2.2

```
input = string FileName, string CaseCategory, list EntityList, list
    RelationList, integer Solution)

if(Solution = 1)
    make new case C with name 'FileName'
end
else if(Solution = 2)
    if(there exist case X with solution 'CaseCategory')
    then
    set case C to X
    else
    make new case C with name 'CaseCategory'
    end
end

for all entities E in EntityList
    add E to the domain model
    add E to case C
end
for all relations R in RelationList
    add R to the domain model
    add R to case C
end

if(CaseCategory equals 'Ukjent')
then
    CaseStatus = 'Unsolved'
else
    CaseStatus = 'Solved'
end
add CaseStatus to case C

output = C
```

Figure 7.7: Pseudocode for Process 2.3

7.5.3 Process 3: Run CBR Session

Figure 7.8 illustrates the data flow in process 3. A CBR session is initiated when the user selects and double-clicks an unsolved case in the GUI. The selected case is then compared against the cases generated in process 2. This is done according to the method described in chapter 5. The category is not directly returned to the user. Instead, the result of the CBR session is presented in the GUI, where the user can clearly see the most similar case.

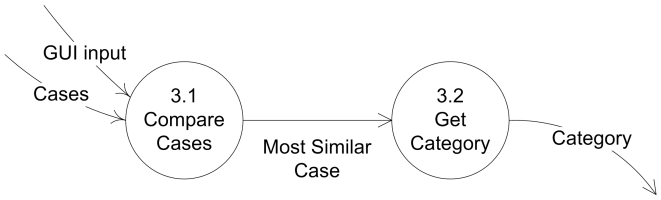

Figure 7.8: Decomposition of Process 3: Run CBR Session

Chapter 8

Implementation

This chapter describes the proof-of-concept implementation of the design proposed in the previous chapter. We will begin with discussing the process of implementation, before presenting the implemented case structure, the case generation and the comparison between textual cases.

8.1 The Implementation Process

The implementation done in this project has been characterized by adjustment and modification of old Creek code rather than making entirely new functions. Most of the functions needed in this project have only been available in TrollCreek code, so they have been modified to fit to the new code structure.

We soon discovered that the relations extracted by CoglibTest were less suitable for reasoning with than first expected. Since the extracted relations was unfitted as causal relations, we ended up with not using any of the knowledge present in the ontology. Because of this, we made the decision to also implement solution 2. This solution is an effort to use the domain knowledge that arises when many cases with the same category are imported.

At the starting point of the project, it seemed as the light weight ontologies could be directly transferable to the case structure used in Creek. However, the entire case structure was radically changed in March, and the new structure, probably superior to the old in many ways, was not as intuitive as the old structure with respect to network cases. Since the cases are not supposed to have DAG properties, it was difficult to utilize the knowledge given by the RDF representation.

Another characteristic of this implementation is that the code changes has been distributed over a big number of classes and packages, from GUI-classes to reasoning and representation. Both Creek and TrollCreek code has been used and combined. This makes the implementation a bit difficult to follow, but we will still try to present a brief overview.

8.2 Case Representation

Remember from section 5.2.2 that Creek cases are structured as graphs with directions and no cycles. All the edges in the RDF format are directed via one of the 'related to' relations, but we have no guarantee that the graph is free of cycles. Because of this, the concepts in the graph

can not be directly inserted as as case features. Rather, the entire graph must considered an entry of a flat case.

The RDF graphs have the functionality of Concept Maps, which have already been implemented in TrollCreek's *Partition* class. A partition is a subset of the ontology that consists of a selection of entities and relations. Every relation/entity that exist in the partition must also exist in the general ontology.

To add a labeled graph to a case, the clue is to add the entities and relations to a *partition*, and then add this partition as a case entry. Figure 8.1 shows a textual case where this has been done. The partition object, illustrated as a graph with concepts and relations, can be seen in figure 8.2.

	Parameter Type	Value	Unit of Measurement
Section 1	Solution	Solved	string
	Category	Klima	string
	Partition	**Partition Object**	partition

Figure 8.1: RDF Case Represented in Creek

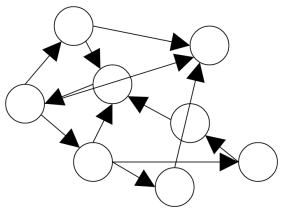

Figure 8.2: A Partition Object

8.3 Adding Text Cases

8.3.1 Import RDF File

When the user selects 'Import XML' from the GUI, an $XMLimportAction$ is created. In the original code, this class makes a new knowledge model, which erases any knowledge already added by the user. We have modified the class so that instead of making a new model, it is first checked whether there exist one in use.

If a knowledge model exists, all concepts and relations expressed by the RDF file will be added to this model. If not, the user is asked through a GUI window whether to use solution 1 or solution 2, and a new knowledge model is created. This ensures that several files can be imported into one knowledge model.

At this point, the responsibility is handed over to the $ImportExportsWizards$, which opens a window that lets the user select the import format. Here, we have just added functionality to choose RDF as well as OWL and XML. A screenshot of this wizard is shown in figure 8.3.

When selecting 'Cognit RDF' in the wizard, an $RDFimport$ is created. This class is new, but its functionality is equal to the '$OWLimport$ class except that it initiates an $RDFimportParser$ rather than an $OWLimportParser$.

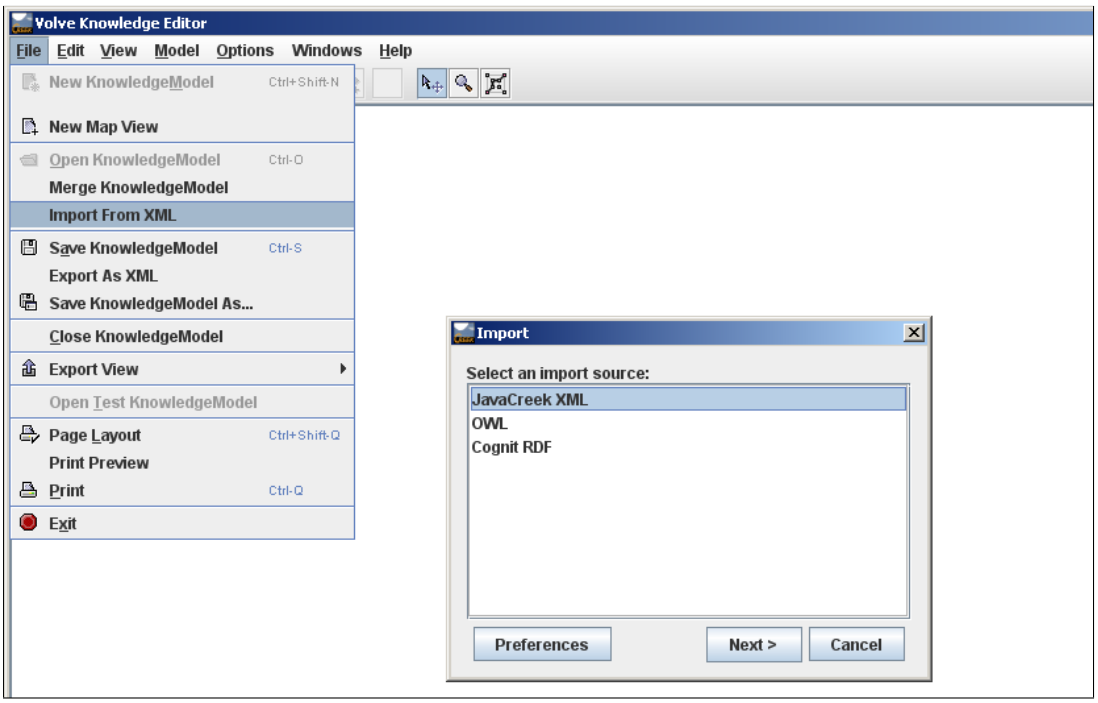

Figure 8.3: Screenshot from Creek: The XML Import Wizard

The structure of the *RDFimportParser* class has been copied from *OWLimportParser*, but all the functionality is new. This class is responsible for RDF analysis as well as transforming the analyzed file to Creek cases. *RDFimportParser* has been modified so that it can implement RDF rather than OWL. In addition, *OWLimportParser* is a TrollCreek class, and it had to be adjusted to generate Creek cases.

Another difference is that in the old code, only some of the OWL statements actually concerned the case. In this implementation, however, the entire file shall be added into a case partition as well as the ontology.

8.3.2 Analyze RDF Statements

The main classes used in this part are *RDFimportParser*, *KnowledgeModelImportUtilities* and *KnowledgeModel*. *KnowledgeModel* is the class that holds all program state information such as the entities and relations that are contained in the ontology. *KnowledgeModelImportUtilities* is the interface between *KnowledgeModel* and the import classes, which helps adding entities and relations.

For the purpose of analyzing the RDF file, we used the Jena framework, which is designed for building Semantic Web applications. Jena is an API in the Java programming language that helps the user create, interpret and manipulate RDF graphs. We have used two of the functions that is delivered by Jena. First, the entire RDF file is read trough and divided into statements. Second, Jena analyzes of each statement and outputs the subject, predicate and object. This delegation process is illustrated in figure 8.4.

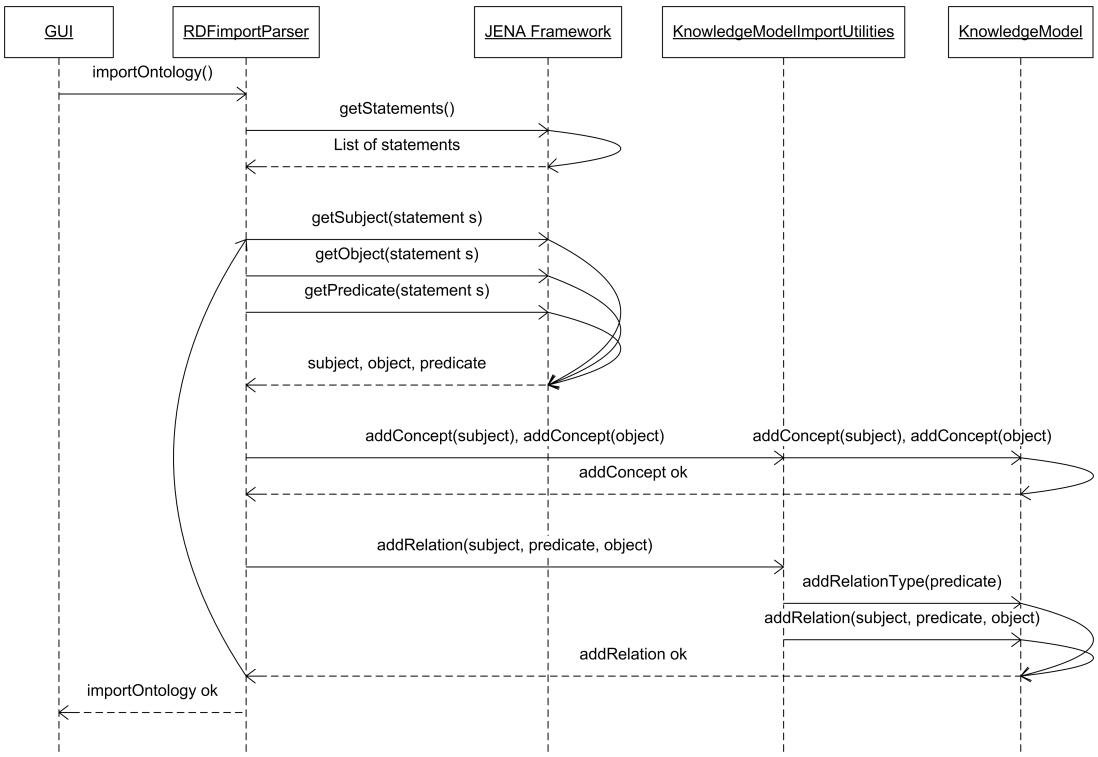

Figure 8.4: RDF Import Sequence

8.3.3 Generate Creek Case

As Jena analyzes a statements into concepts and relations, $KnowledgeModelImportUtilities$ adds these concepts and relations to the general ontology. The relation type must be defined as an entity in the knowledge model before a relation can be added. Therefore, *related to*, *weakly related to* and *strongly related to* have been predefined in the class $SeparatedCaseModel$, which contains all entities and relations of the general ontology.

When adding entities to the ontology, these must be integrated with the general domain knowledge to be used for reasoning later. This is done by letting all created entities be subclass of *Attribute* and instance of *Symbol*, two of the entities defined in the $SeparatedCaseModel$.

After the whole RDF file is analyzed, the entities and relations must be added to a $SeparatedCase$. Ignoring the section aspect, we have added the features directly to the case rather than to a section. This have been possible because $SeparatedCase$ inherits from the $CaseSection$ class.

If solution 1 is chosen, $RDFimportParser$ generates a new $SeparatedCase$ as well as a new $Partition$. In solution 2 has been chosen, Creek searches for a case that has the same name as the category of the incoming text. If this is not found, a new case and partition is created.

All concepts and relations are then added to the partition. This functionality is contained in the $Partition$ class. Then, the $Partition\ p$ is added to the $SeparatedCase\ c$ with the following code:

$$\begin{aligned} \text{Entity } e \;=\; & \text{new Entity(km, "Partition: " + } filename\text{, null);} \\ & e\text{.setEntityObject(}p\text{);} \\ & c\text{.addEntry(}p\text{, " ", }e\text{);} \end{aligned}$$

If the case category equals 'Ukjent', there will be added a string entry to the case, with the name 'CaseStatus' and the value 'Unsolved'. Else, it will be set to solved, and the category is added as a case entry as illustrated in figure 8.1. Then, the name of the *SeparatedCase* is changed, either to the same as the file name (in solution 1) or the same as the category (in solution 2). This is because categories can not be used as case names in solution 1, as each case must have an unique name. File names are inappropriate as case names in solution 2, because several files are combined to form a case.

8.4 Case Matching

When the user double-clicks an unsolved case in the GUI, a CBR process is run automatically. The unsolved case is then compared against all texts in the database. First, a $CaseComparison$ is evoked. This returns the similarity between two input cases A and B. The $CaseComparison$ is the sum of all $SectionComparison$s, which in turn is composite of $EntityComparison$s.

When comparing two entries, the parameter type is checked. If both case A and case B has an entry with parameter type T, the similarity between these entries are calculated. We had to add *Partition* as parameter type, so that when two partitions are compared, a $PartitionComparison$ is evoked. A $PartitionComparison$ inherits functionality from $Comparison$ and compares two entities $e1$ and $e2$.

$PartitionComparison$ returns the similarity between the entities by extracting the partition. The partition represented by entity e can be extracted from the entity with the method $e.getEntityObject()$.

The $PartitionComparison$ returns the Jaccard Coeffisient between the partitions $p1$ and $p2$. To help calculate this measure, there is generated four $SubModel$s:

- The union of all entities in $p1$ and $p2$
- The union of all relations in $p1$ and $p2$
- The shared entities between $p1$ and $p2$
- The shared relations between $p1$ and $p2$

The Jaccard Coeffisient is calculated by dividing the size of the shared models on the size of the union models, as described in chapter 7. This formula was already implemented in the $PartitionComparison$ class, but did not yield the correct answer and had to be re-implemented.

In addition, we have done minor changes to several other classes such as $PartitionAttribute$, $DefaultComparisonController$ and $PartitionSimilarityMeasure$, mainly because they used inherited functionality that did not correspond to the new Creek code.

When all entry similarity are calculated, they are weighted and added to obtain section similarity, and finally, case similarity. This process is then repeated between the unsolved case

and all the solved cases, and the result is presented to the GUI. This is unchanged from the original program.

The modified classes mentioned above can be found in the following Creek and TrollCreek packages:

jcreek.cke.importexport.XMLimportAction
jcreek.cke.importexport.ImportExportsWizards
jcreek.cke.importexport.rdf.RDFimport
jcreek.representation.importexport.rdf.RDFimportParser
jcreek.representation.importexport.KnowledgeModelImportUtilities
jcreek.representation.KnowledgeModel
jcreek.representation.SeparatedCaseModel
jcreek.representation.SeparatedCase
jcreek.representation.CaseSection
jcreek.representation.Partition
jcreek.reasoning.PartitionComparison
jcreek.representation.entitytype.PartitionAttribute,
jcreek.reasoning.PartitionComparison.PartitionSimilarityMeasure
volve.cbr.DefaultComparisonController

Part V

Analysis

Chapter 9

Proof-of-Concept Testing

To test whether this system is superior to comparable systems on a real-life domain, numerous tests would have to be carried out to make sure the results are statistically valid. To perform one test, each single text must be inserted into CoglibTest, copied and temporarily saved, manually modified, saved again and then finally being imported into Creek. Obviously, a large-scale testing would have been a too extensive task giving the time limitations. Therefore, we have designed some basic proof-of-concept tests which demonstrate the general functionality of the system.

In all the following tests, we have fed the framed text(s) into CoglibTest and modified the output according the method described in chapter 6 to make it well-formed RDF. After that, the output has been saved and imported to Creek.

9.1 Test 1: Run a general CBR session

In test 1, we want to find out whether there is possible to run a CBR session that categorizes an unknown text. We have chosen input where the category names do not appear in the texts. The ability to categorize such input is a requirement from section 1.4.

We have chosen the following input:

Text 1

Ukjent Kvadratmeterprisen for leiligheten blir på over 100.000 kroner. Selgeren kjøpte leiligheten i 1997 for åtte millioner kroner. Finansavisen beskriver salget som Norges mest innbringende leilighetssalg.

Text 2

Klima Det koster lite å begrense temperaturøkningen i verden til to grader, mener FNs klimapanel. Prislappen er ca. 0,12 prosent av verdens BNP (brutto nasjonalprodukt).

Text 3

Bolig Leiligheten, som går over to etasjer, skal ikke ha vært til salgs, men da megleren på vegne av en klient ringte og fristet med millionene, slo eieren til.

Text 1 is uncategorized, but the content indicates that it should be categorized under the heading 'Bolig'. We therefore expect Text 1 to be more similar to Text 2 than to Text 3.

Results of Test 1

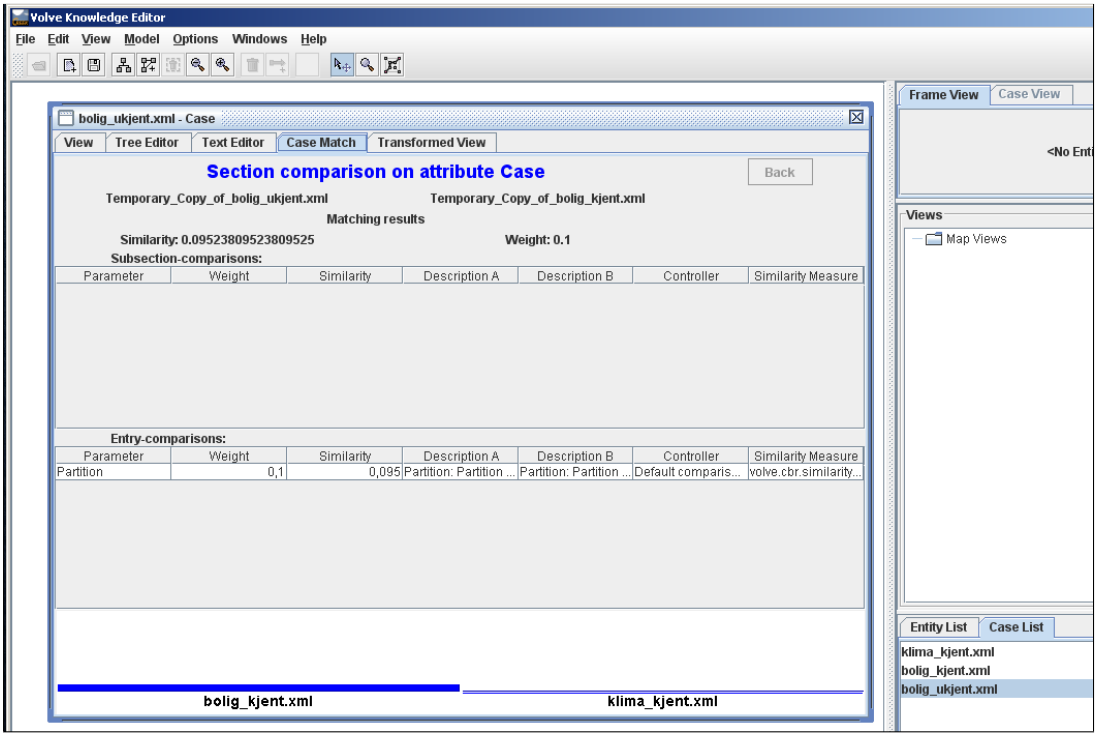

Figure 9.1: Functionality Test 1: Run CBR Session

The result of this test can be seen in picture 9.1. In the bottom right corner we can observe the case list. There has been generated three cases, which all have the same name as the corresponding RDF file. The unknown test is correctly classified under the heading 'Bolig'. The similarity between Text 1 and Text 3 has been calculated to be 9,52%. This is a sensible number, considering that the two texts share a few words and probably also a few RDF relations. Text 2 is not at all similar, which is exactly as we expected.

9.2 Test 2: Visualize an RDF File

In test 2, we test whether there is possible to visualize the concepts of a text in the graphical users interface. We have imported the following text:

Bolig Leiligheten, som går over to etasjer, skal ikke ha vært til salgs, men da megleren på vegne av en klient ringte og fristet med millionene, slo eieren til.

Results of Test 2

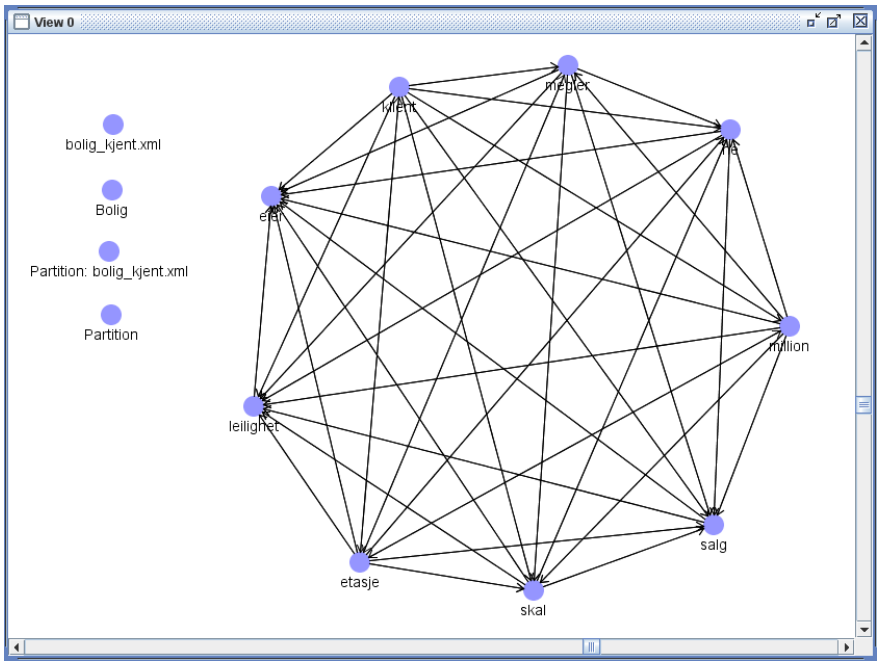

Figure 9.2: Functionality Test 2: Visualize an RDF File

The result has been demonstrated in figure 9.2, which shows a map view of all the concepts that are extracted from the framed text. From this text, only *related to* relations have been extracted, probably because it is short and contain no proper nouns or phrases. An arrow from a concept to another means that the origin concept is related to the target concept.

9.3 Test 3: Combine Cases in the Domain Model

In this implementation, cases have the form as graphs consisting of entities and relations. When two cases contain the same entities, they are merged together in the domain model. The purpose of this test is to check whether the knowledge expressed in two different cases is really integrated.

The test is performed by using the same texts as in the first test, loading them and opening a map view on one of the common nodes.

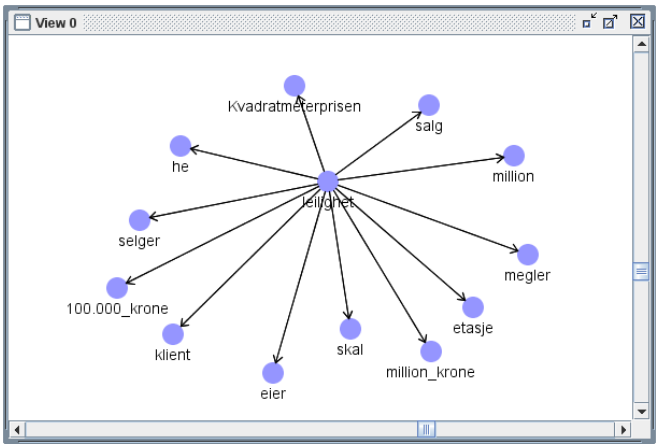

Figure 9.3: Functionality Test 3: Combine Cases in the Domain Model

Results of Test 3

The result has been demonstrated in figure 9.3, which shows a map view on the node 'leilighet'. We observe that this concept appears in both texts (although in the grammatical tense 'leiligheten'). From the node view we can see that this node is now related to concepts that appear in different text cases. For example, 'leilighet' has a relation to both 'selger' (mentioned in Text 1) and to 'etasje' (Text 3).

9.4 Test 4: Recognize a Duplicate

In this test, Text 2 is a duplicate of Text 1, and Text 3 is added as a control case. We expect very high similarity between the first two texts.

Text 1

Ukjent
Kvadratmeterprisen for leiligheten blir på over 100.000 kroner. Selgeren kjøpte leiligheten i 1997 for åtte millioner kroner. Finansavisen beskriver salget som Norges mest innbringende leilighetssalg.

Text 2

Bolig
Kvadratmeterprisen for leiligheten blir på over 100.000 kroner. Selgeren kjøpte leiligheten i 1997 for åtte millioner kroner. Finansavisen beskriver salget som Norges mest innbringende leilighetssalg.

Text 3

Klima
Det koster lite å begrense temperaturøkningen i verden til to grader, mener FNs klimapanel. Prislappen er ca. 0,12 prosent av verdens BNP (brutto nasjonalprodukt).

Results of Test 4

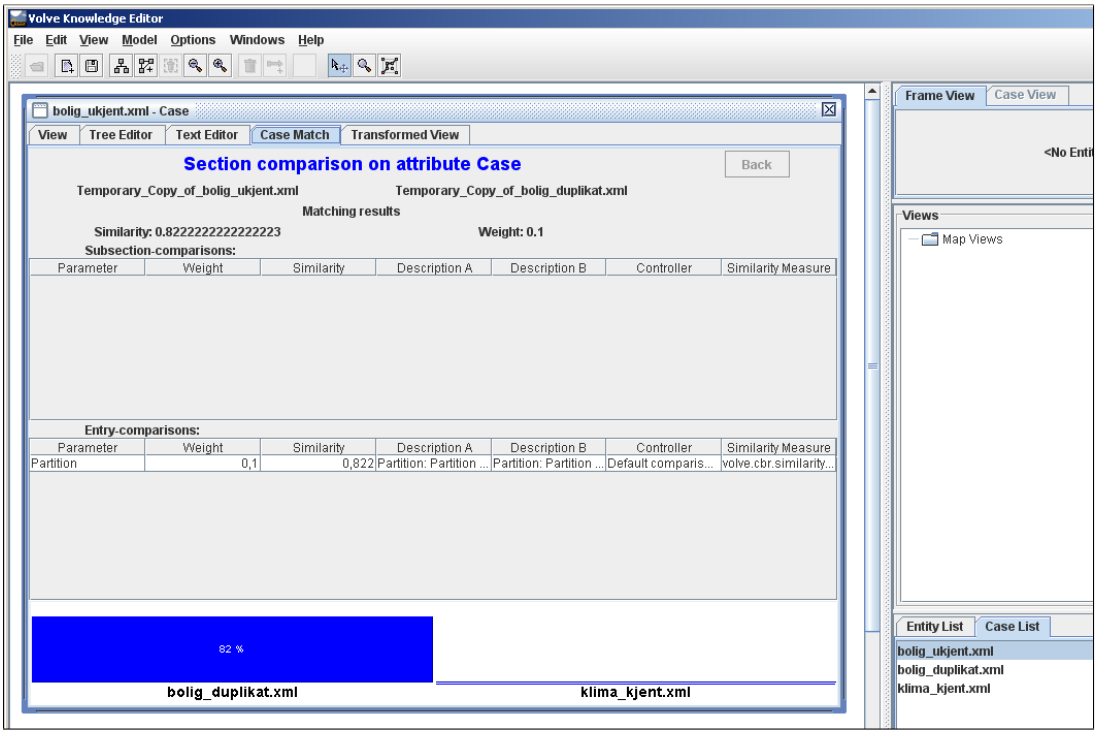

Figure 9.4: Functionality Test 4: Recognize a Duplicate

The similarity between the original and the duplicate is calculated to be 82.22%. This can be checked with figure 9.4. This is lower than we expected, which would of course be 100%. The reason for this has to do with how the section comparison is implemented in Creek. Remember that an unsolved case has only one section with two entries, namely the partition and the case status. The duplicate, on the other hand, has one section with three entries: The partition, case status and the category name. Since the section are compared directly, the cases will not be 100% equal.

As long as the similarity between two general cases will never exceed the similarity obtained by comparing two duplicates, the exact percentage is not really important. The next test is designed to control that the similarity decreases when modifying the duplicate slightly.

9.5 Test 5: Discover Changes to the Duplicate

This test is performed by using the same texts from Test 4, but two of the 25 RDF statements have been removed from the duplicate RDF file and added to the different RDF file.

We expect the similarity of the duplicate to be slightly reduced, and the non-duplicate to become slightly more similar.

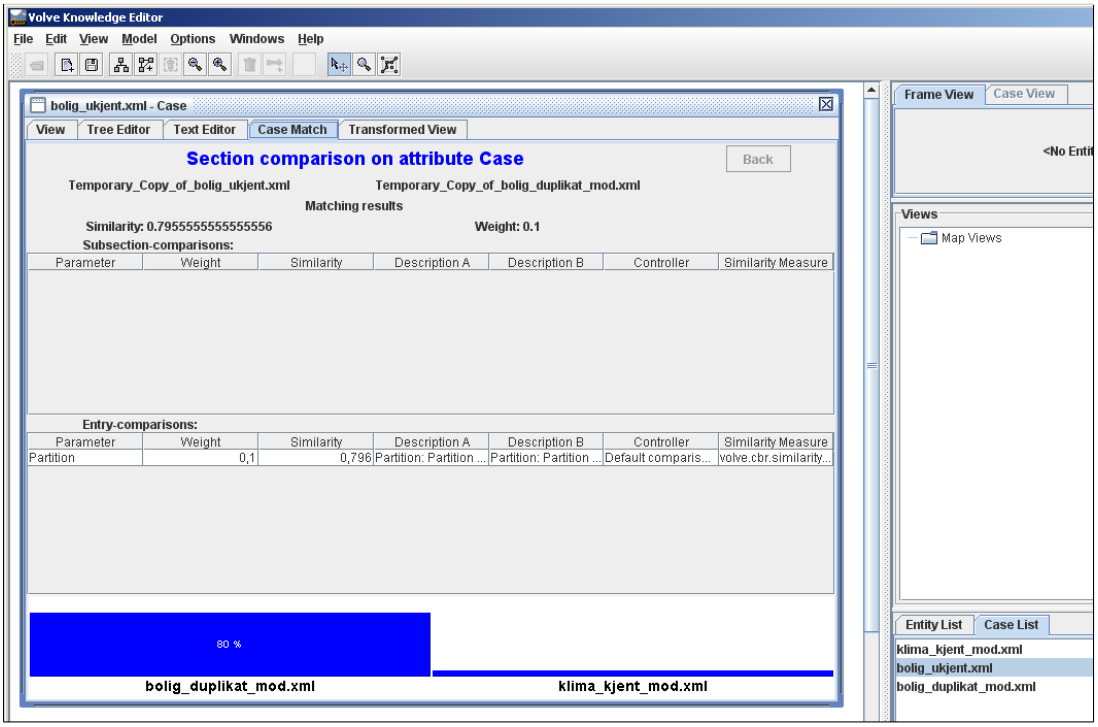

Figure 9.5: Functionality Test 5: Discover Changes to Duplicate

Results of Test 5

Please compare figure 9.4 and 9.5. As can be seen in figure 9.5, the modified duplicate has now a 79.55% similarity with the unsolved case, which is 2.67% less than the original duplicate. Though the GUI only visualize the percentage of the best case, we can see by comparing figure 9.4 and 9.5 that Text 3 has gotten slightly more similar.

9.6 Test 6: Recognize a Duplicate Partition

The last test is performed on the implemented solution 2. The functionality in solution 2 is exactly the same as in solution 1, except when encountering two cases that have the same category.

In this test, we have divided the input case into two smaller texts, analyzed each of them with CORPORUM, and imported both into Creek. Obviously, they share the same category, and will therefore be added to the same case.

The purpose if this test is to check whether we can recognize an entire category if this is similar to the input text. We expect that the case obtained by adding these text will be very similar to the unsolved case.

Text 1

Ukjent
Kvadratmeterprisen for leiligheten blir på over 100.000 kroner. Selgeren kjøpte leiligheten i 1997 for åtte millioner kroner. Finansavisen beskriver salget som Norges mest innbringende leilighetssalg.

Text 2

Bolig
Kvadratmeterprisen for leiligheten blir på over 100.000 kroner. Selgeren kjøpte leiligheten i 1997 for åtte millioner kroner.

Text 3

Bolig
Finansavisen beskriver salget som Norges mest innbringende leilighetssalg.

Text 4

Klima
Det koster lite å begrense temperaturøkningen i verden til to grader, mener FNs klimapanel. Prislappen er ca. 0,12 prosent av verdens BNP (brutto nasjonalprodukt).

Results of Test 6

First, observe from figure 9.6 that the cases are now named after the domains and not the text files.

Second, the similarity percentage between the unsolved case and the 'Bolig' case is 79.17%. As expected, this is very close to the duplicate in test 3. The reason why the number is somewhat lower is that some statements are created across sentences, as we described in chapter 6. Text 1 contains more information than the combined information extracted in Text 2 and Text 3.

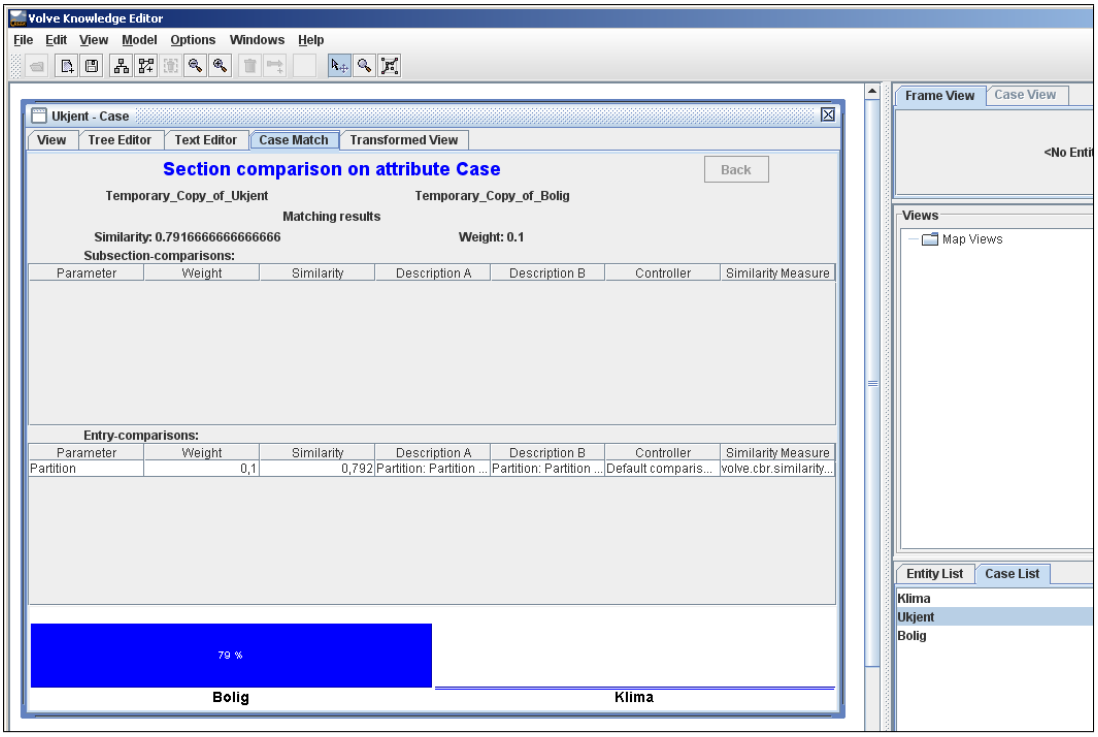

Figure 9.6: Functionality Test 6: Recognize a Duplicate Partition

Chapter 10

Discussion and Concluding Remarks

10.1 Discussion

Because of the limited project scope, we have not carried out an extensive testing that would ensure that the system can be run on a practical situation. Instead, we have tested the implemented program on a controlled domain, where texts are equal in length, and the categories ('bolig' and 'klima') are very dissimilar and therefore easy to differentiate by the system. Some of the words in the input text have also occurred in the most similar output text. In all the tests we have run, we have obtained good results that demonstrate the required functionality. The question is, however, whether or not these results could have been extended to a more realistic situation?

Since the similarity measure is dependent on the number of overlapping concepts and relations, we know that the system will not recognize a text which have none of the word in common with the unknown text, even if it can be argued that these share some semantic structure which make them similar. However, neither can any word frequency based program do this, since they would inevitably be dependent on the number of common word. To make recognition on purely semantic basis possible, we would have to use an ontology that is separate from the cases, where synonymy and relation information can be found.

All the text we have used for testing the implemented program have been of a similar length. On a practical situation, texts will usually be much longer, and their lengths might be unequal. We believe, however, that the program will perform equally well if the texts have different length, as long as we assume that the proportion of common words is independent of the text length. Even though a longer texts will generate bigger RDF graphs, the number of entities and relations in common with the input case will also be higher, and the Jaccard Coefficient will calculate this proportion.

10.1.1 Knowledge Containers in Implemented System

In the program we have implemented, knowledge is represented in the following three containers, which will be discussed in more details:

- The cases (text analysis and case generation)
- The calculation of case similarity
- The domain model

The Case Representation and Transformation

In this project, the texts have been represented in a network structure. This representation is very knowledge-intensive, because both the important concepts and the relations between them are considered. If the purpose is to get a complete representation of a text, graphs are a very good choice.

The analysis of text have been implemented in CORPORUM's CoglibTest, and we have not had the opportunity to influence the results of this process. CORPORUM uses a layered approach similar to that recommended for Textual CBR by M. Lenz [Lenz 1998]. It is difficult to objectively evaluate the performance of CORPORUM when only considering a handful of texts. It seems, though, that the relations extracted by Kernel 2 are not extremely useful for our purpose. The relations extracted from CORPORUM was briefly evaluated in chapter 6, and in section 6.4, S. Thomassen reached more or less the same conclusions as we did.

The cases we have used in this project consist of two parts, the problem and the solution. We made the choice to transform the CORPORUM's RDF output directly into problem part of the Creek cases. This was the most obvious solution, but we could also have processed the graph before transforming it into a case. For example, we could have used only the key concepts, rather than the relations. However, this would reduce the degree of knowledge of the case representation significantly. Alternatively, we could select only some relation types, for example only the *strongly related to* relations, as these are supposedly more content bearing.

The Calculation of Case Similarity

In this project, we have been reasoning with texts represented as graphs. As similarity measure, we used the modified Jaccard Coefficient in equation 10.1 for this purpose:

$$sim(M_1, M_2, c) = c\frac{|A_1 \cap A_2|}{|A_1 \cup A_2|} + (1-c)\frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \tag{10.1}$$

A is the relations, V is the entities and $c \in [0, 1]$, where we used $c = 2/3$.

Comparing the portion of overlapping key concept is a based on the assumption that similar documents will probably have many words in common. However, how do we know that they will have a high number of similar relations? Once again, consider figure 6.2, 6.3, and 6.4. We see very few cases where the presence of two equal relations would predict similarity better than just the presence of the involved entities. Since phrases, such as 'global_oppvarming' has been discovered and is stored as an entity in the domain model, the relation 'global - weakly related to - global_oppvarming' is unnecessary in reasoning scenarios.

When two relations are compared, they are only matched if the relation type is the same. This means that two concepts linked by the *strongly related to* will not match the same two concepts linked by the *weakly related to* relation. Obviously, a partial match would have been more intelligent than no match. We could instead use only one relation, *related to*, together with a numeric weight that gives the degree of the relation.

Human being see similarity as increasing with addition of common features and deletion of distinctive features [Tversky 1977]. In other words, similarity measures based on set theory are good when imitating human problem solving. However, we we could of course have used another set theoretic similarity measure, for example the Overlap Coefficient from equation 2.4. This measure considers two measures: the number of common attributes and the size of the smallest set. We believe, however, that this measure could lead to misleading results, since it does not consider the fact that a very long text will probably have more words in common with the unknown text, even if they do not belong under the same heading.

Another aspect that should be considered is the fact that a CBR session always returns the category of the most similar text, as specified in the requirement specification. However, most classification systems deploy a lower limit for categorization. When the result is below this threshold value, the document is considered too dissimilar to fit in any of the predefined categories. In the first functionality test in chapter 9, the most similar document share only 9% of the entities and relations in the input case. It can be questioned whether this percentage is too low to qualify for categorization. Note, however, that the duplicate is only 80% similar, so if the results were normalized, the test result would have been higher.

The General Domain Model

The general domain model in Creek is an ontology where all information in all the different cases are specified. The concepts that appear in more than one text, have extended representations. The generated ontology is very useful for representing the information contained in an entire domain. However, the ontology have not been exploited to its full potential in this project, because it is not used for reasoning. The relations within each case is used for reasoning, but there are no synergy between cases that could allow them to use each other's knowledge.

We actually considered letting the relations extracted by CORPORUM inherit the causal property, so that a semantic transformation step would have been included. However, this was dropped because the extracted relations had little in common with transitive relations. Moreover, the relations tend to occur in clusters. Within the different clusters, most entities are already related to each other, so almost no new properties would have been deduced. This can be verified by looking at figure 6.2, 6.3 and 6.4 in chapter 6.

10.1.2 Alternative Exploitation of Knowledge Containers

The intended use of Creek is that more knowledge is represented in the domain model, and less is represented in the cases. In fact, we believe that this solution is perhaps better than the one that was chosen in this project. A case might just as well be only a lists of key words, and the relations between these key words can be stored in the ontology. This solution would make it easier for one case to use the knowledge expressed in another case. The ontology knowledge could, for example, be used for in case transformation, exactly like is done in Creek today. The success of Creek is a good indication of the benefits of this solution.

As mentioned in section 5.4, the Creek's new case structure is not ideal for storing BOW texts, because the parameter types within each section must be unique. The old case structure is more flexible, and we think that this structure could have been useful for a BOW representation of text. This would allow the cases and domain knowledge to be linked more closely. Since the relations extracted from CORPORUM were no near being causal, and Creek's new case structure is unfitted for this, this solution would not have been applicable to this project. However, it should be considered for future research.

10.1.3 The Second Solution

We tested solution 2 in test 6, and the results from this test is promising. However, in this scenario, the entities and relations in the input case are similar to the entities and relations obtained by adding all texts corresponding to a category. This is unlikely to happen in practical situations, where the general ontology is probably many times bigger than the unsolved case. However, we argue that this is not important, as long as the categories are of similar size. Even if one category is bigger than the other, this should have no effect on the result when the Jaccard Coeffisient is used, as explained in section 10.1.1.

Solution 2 is not widely used in CBR, it is rather a kind of category-based reasoning. A related approach can be found in the exemplar-based reasoning framework PROTOS [Bareiss 1989]. Here, similar experiences are combined to form *exemplars*, which are used as basis for categorization. Even if this approach is seldom used in CBR, we believe that it is very useful for ontology building. The most common is building one ontology that contain all information about a domain. In the implmeented solution 2, this is done in the general domain model, but the partitions will contain the category information. The category will grow for each case that is added.

Because of the polysemy that naturally occurs in natural language texts, the same word might appear several times while describing different concepts. Combining knowledge is useful because you get a clustering effect where more information can be deduced. However, there is always a risk that different concepts might be represented as one. The result is that totally unrelated concepts will appear to have some semantic link. When merging after category, this problem might be reduced because there are no longer any intervention between separate domains.

10.2 Achievement of Goals

As mentioned in chapter I, the goal consists of several subgoals. The first was to study Case-Based Reasoning and Information Extraction, Creek and CORPORUM. This has been done thoroughly in the preliminary study of part II and III.

Subgoal #2 was to design a system for textual Case-Based Reasoning that uses CORPORUM's light weight ontologies as input and Creek as the classification tool. The functional design has been done in chapter 7. The implementation in subgoal #3 has been done by extending and modifying the Creek framework. The implementation process and details have been described in chapter 8, and can be verified by inspecting the code and running the program.

The fourth subgoal was to test and evaluate the system. This has been hard to reach, because of our initial plan to test our program in comparison to the master thesis written in 2005 by Erik Rogstad and Øystein Ulset [Rogstad and Ulseth 2005] that was described in section 3.1.3. Because of some missing program files and documentation, we were unable to run the program made in this project. Neither could we compare the two approaches by using the same input texts, because the training sets had been deleted.

Because of the time it takes to run one single test, we were also unable to run a full-scale experimental testing. However, the functionality test in chapter 9 demonstrate the general functionality of the system we have implemented. We will once again mention the requirements from the requirement specification:

- The user shall provide no other input than the texts and categories
- Unsolved texts will be assigned one of the categories that has been inserted by the user

- It must be possible to assign a category name that is not mentioned in the text
- Duplicates will obtain a match close to 100%

These requirements have all been reached, and can be verified by checking with the tests in chapter 9.

10.3 Further Research

10.3.1 Creek and Textual CBR

We believe that Creek does not require substantial changes to be suitable for Textual CBR. The success will probably depend on the relations in the general ontology. To actually utilize the general knowledge in case transformation, the relations must be causal, and in order to draw sensible conclusions, the should perhaps be specified by a domain expert. A text analysis tools can extract key word and less consequential relationships, such as the similarity relation between concepts. We suggest that if Creek should be deployed further for Textual CBR, more relations than just formal causal relations should be used in reasoning. For example, knowing what concepts are synonyms is obviously useful in text classification.

We also imagine a future scenario where the sections are used more as intended. One section may contain data such as the title, author, ISBN code and publishing year of the text. This knowledge should not be intervened with semantic information. Since it is possible to add similarity measures for every new parameter type, this can be used intelligently. For examples, the author of a text often includes several persons, so the similarity measure should reflect the number of common authors. The ISBN code, on the other hand, should probably only be matched syntactically.

If it was in fact possible to easily deduce FOPC axioms from text, this would be extremely useful for Creek's domain model. If some extraction tool could manage to discover predicates such as $\forall x(\text{mammal}(x) \rightarrow \text{animal}(x))$, this can easily be transformed to the causal creek relation *mammal implies animal*, which can be used for sound deduction. Alternatively, the Lexico-Syntactic Pattern Extraction approach that was explained in section 3.2.3, can extract simple relationships such as the synonym and hyponym relationships from Creek's textual knowledge sources.

10.3.2 CORPORUM and Textual CBR

CognIT already have an algorithm that compares two light weight ontologies, and we can only assume that it outperforms the measure implemented in this project. They also have a visualization module, which we unfortunately were unable to test. Anyways, the Creek's GUI might be useful for CORPORUM, since there is possible to visualize the CBR session. There is also possible to open map views on the different nodes and select which relations to see. When graphs are complex, this is a desirable property.

CORPORUM is deployed in text analysis, but as far as we know, there has not been done any effort in merging the texts to make extended domain models by combining the extracted knowledge. In this project, we have made domain models that span over the entire domain as well as one text category. This might be an interesting idea for CognIT to pursue further.

10.3.3 Further Cooperation

Can CORPORUM and Creek benefit from each others functionality? From CORPORUM's point of view, Creek have some interesting properties that should be looked further into, such as the possibility to visualize the comparison between textual cases and combine texts in an extended domain model.

From Creek's point of view, the relations that are extracted from Kernel 2 are not very useful. However, we know that CORPORUM is working on the subclass relations, and other relations that might have a causal or is-a nature. When this functionality is finished, the situation will appear different. Automatic extraction of relations that can be used in Creek's domain model would significantly reduce the knowledge acquisition needed that is needed to make an ontology today.

10.4 Concluding Remarks

In this project we have made two different types of technology cooperate in categorizing unknown texts. We have used the text analysis tool CORPORUM to generate a structured representation of natural language texts, consisting of key concepts and the relations between them. The knowledge-intensive Case-based Reasoning framework Creek have been extended and modified so that it could reason with the output given by CORPORUM. When testing the implemented system, we have observed that Creek and CORPORUM can cooperate in categorizing documents, even if Creek's format of representing cases is different from the output generated by CORPORUM. Because of differences in relation types between Creek and CORPORUM, the general domain knowledge of Creek was not fully utilized during case matching. However, Creek might benefit greatly from using a text analysis tool such as CORPORUM for ontology building.

Bibliography

Aamodt, A. (2004). Knowledge-intensive case-based reasoning in creek. *Funk and Gonzáles-Calero, (eds.) 7th European Conference on Case-Based Reasoning, Madrid, Spain.*

Aamodt, A. and Plaza, E. (1994). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, pages 33–59.

Agichtein, E. and Gravano, L. (2000). Snowball: extracting relations from large plain-text collections. *ACM Press*, pages 85–94.

Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. *In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data.*

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* Addison Wesley, ACM Press.

Bareiss, R. (1989). Exemplar-based knowledge acquisition. *Academic Press.*

Barendregt, H. (1981). *The Lambda Calculus, its Syntax and Semantics.* North-Holland.

Barwise, J. (1977). *An Introduction to First-Order Logic.* North-Holland.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American, Inc.*

Brill, E. (1992). A simple rule-based part of speech tagger. *Proceedings, Third Conference on Applied Natural Language Processing.*

Brüninghaus, S. and Ashley, K. D. (2001). The role of information extraction for textual cbr. *Case-Based Reasoning Research and Development, Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR-01) (Lecture Notes in Artificial Intelligence, 2080)*, pages 74–89.

Brüninghaus, S. and Ashley, K. D. (2005). Reasoning with textual cases. *Case-Based Reasoning Research and Development, Proceedings of the Sixth International Conference on Case-Based Reasoning (ICCBR-05) (Lecture Notes in Artificial Intelligence, 3620)*, pages 137–151.

Champin, P. and Solnon, C. (2003). Measuring the similarity of labeled graphs. *Case-Based Reasoning Research and Development: Proceedings of ICCBR 2003*, pages 80–95.

Cowie, J. and Lehnert, W. (1996). Information extraction. *Communications of the ACM*, 39(1):80–91.

Cunningham, C., Weber, R., Proctor, J., Fowler, C., and Murphy, M. (2004). Investigating graphs in textual case-based reasoning. *Proceedings of the 7th European Conference of Case-Based Reasoning.*

Davis, G., Wiratunga, N., Taylor, B., , and Craw, S. (2003). Matching smarthouse technology to needs of the elderly and disabled. *Workshop on CBR in the Health Sciences, ICCBR'03*, pages 29–36.

Díaz-Agudo, B. and Gonzáles-Calero, A. (2001). A Declarative Similarity Framework for Knowledge Intensive CBR. *Case-Based Reasoning Research and Development : 4th International Conference on Case-Based Reasoning, Vancouver, BC, Canada, 2001, Proceedings*.

Deerwester, S., Dumais, S. T., Landauer, T., Furnas, G., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of Documentation*, pages 391–407.

Fellbaum, C. (1998). Wordnet: An electronic lexical database. *MIT Press*.

Gabel, T. and Stahl, A. (2004). Exploiting background knowledge when learning similarity measures. *Proceedings of the 7th European Conference on Case-Based Reasoning*.

Galvez, C. and de Moya-Anegón, F. (2006). An evaluation of conflation accuracy using finite-state transducers. *Journal of Documentation*, 62:328–349.

Gruber, T. (1994). An ontology for engineering mathematics. *Proceedings of Comparison of implemented ontology, ECAI'94 Workshop*, pages 93–104.

Gupta, K. M. and Aha, D. W. (2004). Towards aquiring case indexing taxonomies from text. *Proceedings of the 7th Annual Conference of the International Florida Artificial Intelligence Research Society*, pages 172–177.

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. *Proceedings of COLING-92, Nantes*, pages 23–28.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ: Prentice Hall (Prentice Hall series in artificial intelligence, edited by Stuart Russell and Peter Norvig).

Kobayashi, M. and Takeda, K. (2000). Information retrieval on the web. *ACM Computer Survey*, 32(2):144–173.

Konolige, K. (1994). Using default and causal reasoning in diagnosis. *Annals of Mathematics and Artificial Intelligence*, pages 97–135.

Kroeger, P. (1992). *Phrase Structure and Grammatical Relations in Tagalog*. CSLI, Stanford, California.

Kulyukin, V. and Burke, A. (2002). Mining free text for structure. *Data Mining: Opportunities and Challenges, Idea Group Publishing, Pensylvania*.

Kvarv, G. S. (2007). Association Rules for Automatic Ontology Construction. Master's thesis, The Norwegian University Of Science and Technology.

Lech, T. C., de Smedt, K., and Bremdal, B. A. (2007). The effect of coreference chaining on information systems. Corporum Documentation.

Lenz, M. (1998). Defining knowledge layers for textual case-based reasoning. *Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning*, pages 298–309.

Lenz, M., Bartsch-Spörl, B., Burkhard, H.-D., and Wess, S. (1998). Textual cbr. *CBR technology: From foundations to applications. Berlin: Springer*.

Levesque, H. and Brachman, R. (1985). A fundamental tradeoff in knowledge representation and reasoning. *Levesque, editors, Readings in Knowledge Representation, Morgan Kaufmann, Los Altos, CA*, pages 41–70.

Lider, B. and Mosoiu, A. (2003). Building a metadata-based website. *Boxes and Arrows*.

Liu, H. and Singh, P. (2004a). Commonsense reasoning in and over natural language. *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*.

Liu, H. and Singh, P. (2004b). Conceptnet - a practical commonsense reasoning tool-kit. *BT Technology Journal*.

McGuinness, D. (2002). Ontologies come of age. *The Semantic Web: Why, What, and How, MIT Press 2002*.

Mitchel, T. M. (1997). *Machine Learning*. McGraw-Hill International Editions.

Nilsson (1991). Logic and artificial intelligence. *rtificial Intelligence 47*, pages 31–56.

Pan, J. and Horrocks, I. (2003). Rdfs(fa) and rdf mt: Two semantics for rdfs. *Proceedings of ISWC*, pages 30–46.

Patel-Schneider, P. F. and Fensel, D. (2002). Layering the semantic web: Problems and directions. *First International Semantix Web Conference*.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). Large margin dags for multiclass classification. *Advances in Neural Information Processing Systems. Cambridge, MA: MIT Press*, 12:547–553.

Recio, J. A., Díaz-Agudo, B., Gómez-Martín, M. A., and Wiratunga, N. (2005). Extending jcolibri for textual cbr. *H. Muñoz-Avila and F. Ricci, editors, Proceedings of the 6th International Conference on Case-Based Reasoning, ICCBR 2005, volume 3620 of Lecture Notes in Artificial Intelligence*.

Richter, M. (1995). The knowledge contained in similarity measures. *Invited Talk, The First International Conference on Case-Based Reasoning, Sesimbra, Portugal*.

Riloff, E. and Lehnert, W. (1994). Information extraction as a basis for high-precision text classification. *ACM Transactions on Information Systems*, 12(3):296–333.

Rogstad, E. and Ulseth, y. (2005). Classification of text documents. Master's thesis, The Norwegian University Of Science and Technology.

Salton, G. and McGill, M. J. (1983). Introduction to modern information retrieval. *New York: McGraw-Hill*.

Sørmo, F. (2007). *Case-Based Tutoring with Concept Maps*. PhD thesis, The Norwegian University Of Science and Technology.

Tomassen, S. L. (2002). Semi-automatic generation of ontologies of knowledge-intensive CBR. Master's thesis, The Norwegian University Of Science and Technology.

Tversky, A. (1977). Features of similarity. *Psychological Review*, 34:327–352.

Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, 11.

Voutilainen, A. (1995). A syntax-based part-of-speech analyser. *EACL-95*.

Weber, R. O., Ashley, K. D., and Brüninghaus, S. (2006). Textual case-based reasoning. *The Knowledge Engineering Review*, 20:3:255–260.

Wilson, D. and Bradshaw, S. (1999). Cbr textuality. *Proceedings of the fourth UK Case-Based Reasoning Workshop*.

Appendix A

CORPORUM Coglibtest Light Weight Ontology

A.1 Input

The input text:

Bolig Leiligheten, som går over to etasjer, skal ikke ha vært til salgs, men da megleren på vegne av en klient ringte og fristet med millionene, slo eieren til.

Chosen language: Norwegian
Chosen output: getRDF

A.2 Output

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Lightweight Ontology, generated by CMCogLib DLL CMCogLib: 1.0.4.30
 CognIT a.s, Halden, Norway-->
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:dcq="http://purl.org/dc/qualifiers/1.1/"
xmlns:oe="http://ontoserver.cognit.no/otk_rdf/"
xmlns:do="http://www.test.com">

<!-- Begin Dublin Core Based Ontology Metadata -->

<rdf:Description about="">
     <dc:Title>Bolig</dc:Title>
```

```
     <dc:Creator>CMCogLib DLL CMCogLib: 1.0.4.30</dc:Creator>
     <dc:description>Bolig
Leiligheten, som går over to etasjer, skal ikke ha vært til salgs, men
da megleren på vegne av en klient ringte og fristet med millionene,
slo eieren til.

</dc:description>
     <dc:publisher>local workstation</dc:publisher>
      <dc:date>2007-05-18</dc:date>
      <dc:type>text</dc:type>
      <dc:format>text/plain</dc:format>
      <dc:language>en-us</dc:language>
</rdf:Description>

<!-- End Dublin Core Based Ontology Metadata -->

<!-- Begin Ontology Description-->

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:bolig"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:leilighet"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:etasje"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:skal"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:he"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:salg"/>
```

```
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:megler"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:klient"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="http://www.test.com/mytest/index.html">
    <oe:isAbout>
          <rdf:type resource="do:million"/>
    </oe:isAbout>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="leilighet">
   <oe:relatedTo rdf:resource="#eier"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="etasje">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
```

```
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="skal">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="he">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="salg">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="megler">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<rdf:Description rdf:about="klient">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#skal"/>
```

```
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="million">
   <oe:relatedTo rdf:resource="#eier"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#leilighet"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#etasje"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#skal"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#he"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#salg"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#megler"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
   <oe:relatedTo rdf:resource="#klient"/>
</rdf:Description>

<rdf:Description rdf:about="eier">
```

```
   <oe:relatedTo rdf:resource="#million"/>
</rdf:Description>

<!-- End Class Ontology -->
</rdf:RDF>
```