Analyzing Databases with Structured and Unstructured Data using Text Cube and Automatic Taxonomy Extraction

Master's Thesis

at

Graz University of Technology

submitted by

Christian Andrich

Advisor: Christian Gütl

Graz University of Technology

Institute for Information Systems and Computer Media

Co-Advisor: Wei Liu

University of Western Australia

School of Computer Science and Software Engineering

April 28, 2011

Analyse von Datenbanken mit strukturierten und unstrukturierten Daten mit Hilfe von Text Cube und automatischer Extraktion von Taxonomien

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Christian Andrich

Betreuer: Christian Gütl

Technische Universität Graz

Institut für Informationssysteme und Computer Medien

Zweitbetreuer: Wei Liu

University of Western Australia

School of Computer Science and Software Engineering

April 28, 2011

Abstract

As the amount of information that is available in both, public and private databases, is rapidly increasing, it steadily becomes harder, to handle all the data, and actually make use of the information. Additionally, the data that comes as natural language text, like for instance in online forums, news articles or product reviews can also contain valuable information for certain entities, but bear the problem, that they can hardly be handled by humans without any further help, due to the huge quantity, and are also hard to process automatically.

In this work, a system is proposed to deal with the analysis of databases, that contain structured data and also unstructured data, in the form of natural language text. For that purpose, the typical OLAP functionality is utilized, which traditionally deals with structured data, and is extended by the so called Text Cube, developed in Lin et al. (2008), to also make use of natural language data.

However, the Text Cube only defines methods to process available information that is derived from natural language text, but not how to extract this information. Therefore, automatic taxonomy extraction is performed on the textual data of the database that shall be analyzed, to extract additional information that can be utilized.

The database is stored in an RDF datastore, thus making use of semantic web technologies. This design decision supports the development of a system with a general architecture, such that it can freely be extended by adding additional information to the database at any time. Finally, the capabilities of the system are demonstrated by performing the proposed methods on an example dataset using a prototype implementation.

Kurzfassung

Durch den rasanten Anstieg der verfügbaren Information, zum einen in öffentlichen und zum anderen in privaten Datenbanken, wird es zunehmend schwieriger, diese Informationsflut zu bewältigen und auch einen Nutzen aus der verfügbaren Information zu ziehen. Hinzu kommt, dass auch Daten die in Form von natürlichsprachlichem Text verfügbar sind, zum Beispiel in Online-Foren, Zeitungsberichten oder Produktrezensionen, wertvolle Information für bestimmte Interessensgruppen beinhalten können. Jedoch ist es durch die große Menge an Daten für Menschen schwierig die Information zu verarbeiten, und auch die automatische Verarbeitung birgt einige Schwierigkeiten.

In dieser Arbeit wird ein System vorgestellt, das dazu dient Datenbanken zu analysieren, die sowohl strukturierte Daten als auch unstrukturierte Daten, in Form von natürlichsprachlichem Text, enthalten. Dafür wird die bekannte OLAP Technologie eingesetzt, die traditionellerweise dazu dient, strukturierte Daten zu verarbeiten. Diese wird durch den so genannten Text Cube, der in Lin et al. (2008) eingeführt wurde, erweitert, um auch Textdaten verarbeiten zu können.

Der Text Cube stellt jedoch nur die Methoden bereit, um Information zu verarbeiten, die aus natürlichsprachlichem Text gewonnen wurde, definiert jedoch nicht, wie man zu diesen Informationen gelangt. Dafür wird eine Methode eingesetzt, um automatisch Taxonomien aus vorhandenen Daten zu extrahiert, um diese dann als zusätzliche Information zu verwenden.

Die Datenbank wird für diesen Zweck im RDF Format gespeichert, und nützt damit Technologien die für das Semantic Web entwickelt wurden. Ein derartiges Design bietet die Möglichkeit, das System so zu entwickeln, dass die Datenbank jederzeit durch zusätzliche Information erweitert werden kann. Zum Schluss werden mit der Implementierung eines Prototyps unter Benützung einer Beispieldatenbank die Möglichkeiten des vortestellten Systems demonstriert.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

date	(signature)

Contents

List of Figures

1. Introduction

The amount of data available in databases, in both publicly available and internal databases of enterprises or other organizations, turned out to be subject to exponential growth in recent years (Bennett, 2006; International Data Corporation, 2010). Apparently, to handle this amount of data raises a number of technical challenges. One of those challenges is, to actually make use of the available information in some way. Therefore, a number of techniques has been established, to tackle this task, which commonly make up the field of data mining.

One of those techniques is *Online Analytical Processing* (OLAP). It is a well established method to analyze huge multidimensional datasets. The term OLAP has been coined in 1993 by Edgar Codd (Codd et al., 1993), although the principle ideas and methods reach back to 1962 (Pendse, 2007). OLAP is well suited to analyze structured data like for instance sales records which contain information like price, quantity, date and so on.

However, databases can also contain unstructured data, e.g. user comments about products, which contains potentially valuable information, but cannot easily be handled, especially if it comes in huge amounts. Therefore, OLAP systems have to be adapted to be able to make use of such information. To do this, *Text Cube* was introduced in Lin et al. (2008), which is an extension to OLAP, which provides methods to integrate natural language data.

Nevertheless, to make use of data that comes as natural language text, this data has to be processed in advance as it doesn't have a well defined structure. Therefore, to provide the contained information in a way, that it can be utilized by the analyzing methods, the specific information has to be extracted. There are several subfields of natural language processing, which deal with the extraction of information from text documents, or text corpora, which could deliver potentially useful information that can be utilized by a Text Cube. Such methods are for instance sentiment analysis, automatic classification, topic extraction or ontology extraction. An ontology, in computer science, is a set of concepts, which can have arbitrary relations to each other. A taxonomy is a special case of an ontology, where the concepts are only related by *is-a* relations and therefore represents a hierarchy. Thus taxonomy extraction is a subfield of ontology extraction. Taxonomies can be a especially useful if employed in a Text Cube, as hierarchical structures are a crucial component of OLAP.

The goal in this work is, to combine the methods of automatic taxonomy extraction with Text Cube, including an appropriate method to store and access the according information. Therefore the system has to provide the following functionality:

- **Datastore**: A datastore which provides the possibility to access and update the stored information.

- **Natural Language Processor**: A natural language processor, that can access the according information from the datastore, and perform ontology extraction and also simple term extraction. Subsequently, the extracted information has to be written back to the datastore.
- **OLAP System**: A system that provides OLAP functionality including the Text Cube methods. Thus, this module also has to be able to communicate with the datastore to receive the according information.

Structure of the Thesis

In the following three chapters, the fundamental techniques on which the implementation is built upon are discussed. The first is natural language processing in Chapter 2 which gives an introduction to some basics of that field and into the subfields that are relevant for this work.

In Chapter 3 OLAP is introduced. The basic data structure of OLAP and the operations that can be performed with it are described. Furthermore, the Text Cube operations, that extend OLAP such that it can handle textual data is described.

The semantic web technologies that are used for the datastore are described in Chapter 4. These are RDF, which provides a format to describe resources, and OWL, which allows to define the semantics of the data.

Subsequently, in Chapter 5 the requirements for the implementation and a conceptual approach are described, followed by a more detailed description of the implementation of the prototype in chapter 6, where also the results are demonstrated.

2. Natural Language Processing

Natural Language Processing (NLP) has been an active field of research since before 1950 and is today a well established field of computer science with a big variety of subfields. However, it is a field that bears many very difficult problems, and therefore, many of these problems are still unsolved or only partly solved. In this chapter, an introduction to the field of NLP is given including the discussion of some problems and their state of the art solutions. The focus is on those problems that are also relevant for the implementation of this work.

2.1. Fields of NLP

When the filed of natural language processing came up the main focus was on machine translation. The first conference on this topic was held in 1952 at the MIT (Hutchins, 1997). In 1954 a system was demonstrated in the so called Goregetown-IBM experiment, which was capable of translating a very limited number of sentences from Russian to English (Hutchins, 2004). Although researchers then predicted, that automatic machine translation will be a solved problem within five years, it is still an unsolved problem in general today, almost 60 years later.

NLP also includes a whole variety of other subfields like summarization, question answering, natural language generation, speech recognition and many more. Some of these subfields have underlying techniques in common, for instance, many modern approaches to NLP problems are based on machine learning techniques, as will be discussed in Section 2.2. One subfield that will be discussed in more detail in the next Section, is *Text Mining*, as it is the field to which taxonomy extraction belongs to and is therefore of crucial interest in this work. Furthermore, some basic tasks like tokenization and part of speech tagging and approaches to solve them are discussed in Section 2.3, as these are underlying techniques used for the taxonomy extraction in the implementation.

Text Mining

The goal of *Text Mining* is to extract previously unknown information from natural language text (Hearst, 1999; Kroeze et al., 2003). It is related to the field of *Data Mining*, but in contrast to data mining, which deals with structured data, text mining in general deals with data that has no well defined structure.

Text mining can again be divided into subfields like sentiment analysis, document classification and others, according to the data that is expected to be extracted from the available data. Here we want to concentrate on automatic ontology extraction, as it is the most relevant subfield for this work.

Ontologies are fundamental information structures, which can be used for a variety of tasks like for instance by software agents to exchange unambiguous messages (e.g. Dileo et al., 2002) or for document classification (Song et al., 2006). However, it is a very time-consuming and error-prone task to create ontologies by hand and therefore automatic systems, which can extract ontologies from a given set of data are desirable. The according data can in general be any set of documents. Thus in many modern approaches collections of documents from online sources are used as there is a huge amount of data freely available. In Wong (2008) a system is proposed that extracts lightweight (domain) ontologies from given corpora. A lightweight ontology can either be a taxonomy or a thesaurus. The system performs the taxonomy extraction in the following four phases, given a domain corpus and a general corpus:

- **Text Cleaning:** As the documents in the corpora can come from any web source, they are likely to contain noise like spelling errors, abbreviations and improper casings. Thus in the first phase these issues are concerned to create a clean corpus for the next phase. However, if the documents in the corpus come from high quality sources, like scientific papers, this phase can also be omitted.

- **Text Processing:** In this phase, basic natural language processing tasks like sentence parsing and noun phrase chunking are performed. The result of this phase is a list of term candidates, which is the input for the next phase.

- **Term Recognition:** The domain corpus and the general corpus are now used to determine domain terms among the term candidates. Therefore the likelihood that a term is relevant for the domain in question is calculated using the two measures *Termhood* (Wong et al., 2007b) and *Odds of termhood* (Wong et al., 2007a).

- **Relation Acquisition:** To relate terms, online sources namely Wikipedia and Google are used as background knowledge. This is done by calculating the distance between two Wikipedia articles using the categorization system (Wong et al., 2007c) and the normalized google distance (Cilibrasi and Vitanyi, 2007) for any pair of terms. A relation between two terms can then be established via a common parent.

The result of this process is a graph, in which the edges represent a *is-a* relationship between the nodes and is therefore a taxonomy. Although these processing steps are those of a specific system, they demonstrate well, how ontology finding is done in general as all approaches have to perform these steps in some way. However, how this is done, can be very different in different approaches.

Finding Relations of Arbitrary Types

The described system is capable of finding *is-a* relationships. But concepts of an ontology can have any kind of relation between them, not only *is-a*. Therefore, this result is only a first step

towards the goal of automatic ontology extraction. Thus the process of *Relation Acquisition* has to be improved to find more general relations.

For that purpose different methods have already been developed. For instance, the authors in Sanchez and Moreno (2008) use an approach that uses the web as background knowledge and is performed in several steps. This is done in a way, that in each step the result of the previous step is utilized as a base for the current step. Thus firstly, an ontology with only taxonomic relationships is created, as described in Sánchez and Moreno (2006). To find non-taxonomic relationships, the verbs that occur with the identified concepts are investigated. Therefore, the verbs get filtered and classified, to find those which are closely related to the domain of the ontology. The remaining verbs are then used to relate the concepts and thus yielding a more sophisticated ontology.

A different approach to find general relations that uses clustering techniques is demonstrated in Rosenfeld and Feldman (2007). There, relation candidates are represented by tuples $c \in E \times E$ of an entity set E. Each sentence in which an element occurs is said to be a context of that element. Thus the elements of a tuple have a common context if they occur in the same sentence. For the clustering, features of the relation candidates are required, which are derived from surface patterns. Such patterns are generated by using all contexts of all entities, which means, that equal or similar sequences of tokens that occur in a number of contexts yield the patterns for the features. If a number of features has been generated this way, each relation candidate can be associated with a feature vector. The clustering is then performed on these vectors. After the clustering, elements that belong to the same cluster represent the same relation. In the experiments on the NYT95 corpus a precision of 0.930 was achieved with a total of 307 correctly identified relations.

These approaches can yield good results if the relations between the concepts are explicitly contained in the processed corpus. If this is not the case, some useful relations can easily be overlooked. Thus it can be beneficial, to use some further background knowledge to overcome these limitations. In Wong et al. (2009) Wikipedia is used as such a background knowledge. However, the relations found with the described method are again not of any type, but limited to hierarchical, associative and polysemous relations.

2.2. Machine Learning and Statistics in NLP

Most modern approaches of NLP, reaching from low-level to high-level tasks, involve machine learning and statistical methods of some kind. Reasons for that are, that natural language is not well structured, ambiguous and redundant and therefore simple algorithms with fixed rules and grammars are often not well suited to process the data and extract information. Furthermore, in natural language, the same information can be expressed in a myriad of ways, which requires information extraction methods to be very flexible.

Machine Learning Techniques commonly used in NLP

Among others, the following popular machine learning techniques are also widely used in NLP, and are also involved in some of the solutions of the tasks discussed in Section 2.3:

- **Hidden Markov Models (HMM)**: A HMM is used to estimate the probability of a sequence of states, where the sequence is distributed according to a markov process. The states are not directly visible (hidden), but observations which are dependent on the states (by output probabilities) are given. To establish the model, the transition probabilities between the states and the output probabilities have to be calculated. For this purpose, a set of labeled training data is needed. This method is used, for instance, in Shen and Sarkar (2005) for noun phrase chunking or in Brants (2000) for part-of-speech tagging.
- **Supprot Vector Machines (SVM)**: SVMs can be used to assign sample vectors to one of two classes. The classifier is established by finding the hyperplane in the feature space, that best separates the classes according to the samples in the training set, while maximizing the distance between the samples and the hyperplane. The hyperplane is then used to separate the classes. This is used, for instance, in Abacha and Zweigenbaum (2011) for relation extraction or in Joachims (1998) for text classification.
- **Maximum Entropy Classifiers**: With this method, the classifier is established by maximizing the entropy of the underlying probability distribution on the training set. For this purpose, the information in the training set is considered as testable information. This means, that the probability distribution has to be chosen such that it is consistent with this information. This method is used, for instance, in Tsuruoka and Tsujii (2005) for part-of-speech tagging or in Chen et al. (2008) for subjectivity analysis.
- **Clustering**: Clustering is an unsupervised learning technique, as it creates clusters of given items, according to their distance to each other. Thus, items which are close to each other according to some metric are assigned to the same cluster. There are different approaches to find clusters of a given set of items like for instance *k-Means*[1] or the *k-nearest neighbor* algorithm (Fix and Hodges, 1951). Clustering methods are used, for instance, in Rosenfeld and Feldman (2007) for term-relation finding or in Fodeh et al. (2011) for document clustering.

Background Knowledge

Furthermore it can be useful to use background knowledge to improve the accuracy and efficiency of learning and classification algorithms. For instance, in Majewski and Szymanski (2008) this was done for text categorization by involving background knowledge acquired form ConceptNet[2] which provides information in the form of a semantic network. There is also a number of other knowledge bases available, which provide structured data like for instance OpenCyc[3], WordNet[4]

[1] http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html
[2] http://csc.media.mit.edu/conceptnet
[3] http://opencyc.org/
[4] http://wordnet.princeton.edu/

or DBpedia[5]. These knowledge bases can provide information from a variety of domains, however, if such background knowledge can be utilized strongly depends on the application.

2.3. Basic Tasks of NLP

For many tasks in NLP the input data comes as text corpora. A text corpus in general is a collection of any kind of textual documents like for instance scientific papers, news articles, web forum postings or web pages. Apparently, the incoming data has to be processed in some way in order to manipulate it on a higher level in an NLP system. This part of the whole process is usually called text preprocessing and involves tasks like sentence segmentation, tokenization, part-of-speech tagging and others, depending on the corpus and on the data that is needed for further processing. In this Section, approaches to perform the mentioned tasks and also noun phrase chunking are discussed, as they are necessary for the taxonomy extraction in the implementation.

2.3.1. Segmentation and Tokenization

Segmentation in general is the task of separating the input into single units. In NLP a common task of segmentation is to separate the sentences of a given document. This task is not as simple as it might appear at the first glimpse. The basic approach is to look for characters that mark the end of a sentence like ".", "?" or "!". However, apparently these characters do not necessarily indicate the end of a sentence. Especially the period often indicates an abbreviation instead of the end of a sentence, or it can even be both at the same time. Furthermore, multiple periods like "..." do not mean that multiple sentences end at these positions. Because of such ambiguities of delimiting characters, sophisticated methods have been developed to perform the task of sentence splitting. Different approaches involve rule-based classifiers or classifiers that have been trained using supervised or unsupervised methods.

In Clough (2001) a rule based approach, based on Mikheev (1999), that has been implemented in Perl is demonstrated. The goal there was to achieve a high accuracy on a wide range of corpora with an easy to use and modifiable tool. An advantage of rule-based approaches is, that they don't need to be trained on any training set and they can achieve a good performance concerning processing speed. The rules of the particular implementation in Clough (2001) are based on assumptions like:

- Sentences are bounded by one of [.!?]
- Periods that are followed by a whitespace and a capital letter mark sentence boundaries
- Periods followed by a digit do not mark sentence boundaries
- Periods followed by other punctuation marks are probably not sentence boundaries

[5] http://dbpedia.org/

and some more assumptions of that kind. The according rules can then easily implemented. The implementation has been tested on the *British National Corpus* (BNC) and on the Brown Corpus. An accuracy for the correct disambiguation of periods of 98.59% has been achieved on BNC and of 97.61% on the Brown Corpus.

In Kiss and Strunk (2006) an approach is described, that uses an unsupervised technique to detect sentence boundaries. The focus in this approach is to detect if a period marks an abbreviation or not, where if it does, it can still mark the end of a sentence at the same time. The system works in two stages, first the *Type-Based Classification Stage* which yields an initial annotation of the periods, and second the *Token-Based Classification Stage*, which yields the final annotation. In both stages, likelihood ratios are used to detect abbreviations:

- **Type-Based Classification Stage**: In this stage, the likelihood of an period being an abbreviation marker depends on three properties: (i) abbreviations occur with a final period (this assumption is not always true, but in cases where it is wrong, there is no period to disambiguate), which means, that a word that is followed by a period more frequently than expected is likely to be an abbreviation, (ii) abbreviations tend to be short and (iii) many Abbreviations contain additional internal periods. The probabilities for these properties can be calculated by investigating the according occurrences in the corpus. A candidate is then considered as an abbreviation marker, if the product of these measures is greater that a certain threshold.

- **Token-Based Classification Stage**: In this stage, the likelihood of a period being an abbreviation marker is determined by the investigation of the tokens which surround the period. For that purpose, three heuristics are used to improve the results of the previous stage: (i) *The Orthographic Heuristic* investigates if the word after the period is uppercase or lowercase and compares that to other occurrences of that word, and decides if the period marks the end of a sentence based on these observations. (ii) *The Collocation Heuristic* states that a period between two words that form a collocation is likely to mark an abbreviation instead of a sentence boundary. (iii) *The Frequent Sentence Starter Heuristic* investigates how often certain words occur after a sentence boundary according to the result in the previous stage. Thus periods which are followed by words which have a high likelihood of being sentence starters are more likely to be sentence boundaries.

The system has been tested on corpora in several languages and compared to other state-of-the-art systems. Although the results of the system on specific corpora are slightly inferior to those of the best systems known (e.g. an error rate of 1.02% was achieved on the Brown Corpus), the accuracy among different languages and domains is very stable. Furthermore only the corpus itself is required for the classification but no further data like abbreviation lists or training sets.

Tokenization

A further step is to tokenize the sentences into single words, which is again a segmentation problem. Several methods, using a variety of techniques reaching from rule based to statistical, have been developed to solve this task (e.g. Kaplan, 2005; Graña et al., 2002; Grefenstette and Tapanainen,

1994). Like sentence segmentation, the task is not as easy as it seems at the first glimpse, as there is a number of issues that have to be addressed (Schmid, 2008) like for instance:

- **Periods and other punctuation marks**: As punctuation marks like ".", ",", ":", "!" and others can have several meanings, they can either be considered as a token on their own, or as a part of another token.
- **Multi word expressions**: For expressions like "et cetera" or dates like "Sep. 2, 1983" it might be useful to consider them as one token, instead of splitting it up in several tokens.
- **Clitics**: Expressions like "don't" or "I'm" actually consist of two words and should therefore be split accordingly.

How such issues are dealt with can also depend on the application in which the resulting data is used. This also makes it hard to evaluate and compare the results of different tokenization systems, as discussed in Habert et al. (1998). The problem is, that there is no formal basis which describes exactly what a token is. Thus the comparison and evaluation only makes sense, if application specific issues are taken into account.

2.3.2. Part-of-Speech Tagging

In *Part-of-Speech Tagging* (POS-tagging) each word or token is assigned with its corresponding part of speech. The difficulty of this problem lies in the multiple meanings a word can have. For instance, consider the word "hurry" in the sentences "*She is in a hurry.*" and "*She told him to hurry.*". In the first sentence, "hurry" is a noun and in the second sentence, it is a verb, where in both cases it is spelled equally. Thus a POS-tagger has to make a decision, based on where in a sentence the word occurs. A number of methods have been developed to solve this task which are based on techniques like HMMs (e.g. Brants, 2000), SVMs (e.g. Giménez and Márquez, 2004), *Maximum Entropy Classifiers* (e.g. Stanford Tagger[6]) and others. A list of state of the art POS-taggers is maintained at the website of the *Association for Computational Linguistics* (ACL)[7].

Classifiers using Hidden Markov Models

A classifier which uses HMMs for instance, makes its decision based on the likelihood of one certain type of word following another type of word. If we again take the example sentence "*She told him to hurry.*" then the classifier might consider "hurry" as a verb, because the probability that "to" is followed by a verb is higher than that it is followed by a noun.

Applying Tagsets

As a result of the tagging process, each token shall be marked with the according tag in a form like: "*She/PRP told/VBD him/PRP to/TO hurry/VB ./.*", where the tags can vary on different systems. In the example, *PRP* stands for pronoun, *VBD* for past tense verb, *VB* for verb in

[6] http://nlp.stanford.edu/software/tagger.shtml
[7] http://www.aclweb.org/aclwiki/index.php?title=POS_Tagging_%28State_of_the_art%29

base form, *TO* for "to" as preposition and "." for sentence terminator, as defined in the Penn Treebank Tagset[8]. As it can be seen in the example, the tags can differ for different word forms, like singular and plural or present and past tense (*VB* vs. *VBD*) and also punctuation marks are tagged according to their meaning. Thus the number of tags in different tagsets can differ. For instance the tagset of the Brown Corpus contains 82 tags, while the Penn Treebank Tagset contains 45 tags (including punctuation tags). Therefore, like in the case of different tokenization methods, the appropriate method/tagset depends on the application and further processing.

2.3.3. Noun Phrase Chunking

The goal of text chunking is to divide sentences into smaller non-overlapping structures, such that the parts of each structure semantically belong together (Abney, 1991). Thus *Noun Phrase Chunking* (NP-chunking) is a subtask of text chunking.

There is no formal definition of the structure of a noun phrase (NP), but in general, a NP consists of a head noun with some optional modifiers like[9]:

- **Noun phrases**: modifiers that are again NPs, e.g. "Take care with the ***kitchen*** *knife*.", where the noun "kitchen" is part of the NP "kitchen knife".
- **Adjective phrases**: where an adjective phrase modifies the head noun, e.g. "He showed her the ***new*** *laptop*.".
- **Prepositional phrases**: phrases formed by a preposition and a prepositional component following the NP, e.g. "The *girl* ***in the red dress*** looked at you.".
- **Verb phrases**: where the modifier is a verb phrase e.g. "The *man* ***reading the newspaper*** is the boss.".
- **Relative clauses**: subordinate clauses, usually beginning with a relative pronoun as modifiers for the NP, e.g. "He is the *actor* ***who won three oscars***.".

When applying an NP-chunker, usually base NPs are the desired output. These are NPs that do not contain other NPs or NP postmodifiers. Consider the example sentence (from the Brown Corpus):

At [the same time], [he] remains fairly pessimistic about [the outlook] for [imports], given [continued high consumer and capital goods inflows].

Here the NPs are marked by square brackets. If the two NPs "*the outlook*" and "*imports*" are joined as "*the outlook for import*", the result would again be an NP, but not a base NP. In Ramshaw and Marcus (1995), the authors considered text chunking as a labeling task, as the chunks do not intersect and can therefore be labelled accordingly. This notation was adopted by many authors since, and is the common method today.

[8] http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[9] http://www.suite101.com/content/the-grammatical-noun-phrase-modifier-in-english-a107027

State of the Art

Modern approaches for NP-chunking are using a variety of techniques like SVMs, *Conditional Random Fields* or HMMs. Like for POS-taggers, ACL also maintains a list for state of the art NP-chunking methods at its website[10].

2.3.4. Handling Noisy Text Corpora

Many modern natural language processing systems, especially text mining systems, have to deal with data that comes from the internet, like forum postings, emails, blogs or recessions. The advantage of that kind of data is, that there is a huge amount of it available and it can contain information about recent events. However, this comes at the cost of a poor quality of the data, as the writing style on the internet is usually informal and contains spelling errors, improper casings, ad-hoc abbreviations and other flaws of that kind. Thus before processing this data, it is useful to perform some kind of text cleaning during the preprocessing phase.

Cleaning email data

In Tang et al. (2005), the authors investigated the effect of text cleaning of emails which are used for text mining. The analysis of the emails showed, that 98.4% contained noise (errors) that can influence the result of the text mining process. The approach contained non-text filtering (e.g. headers and signatures), paragraph normalization, sentence normalization and word normalization, where the classification into correct/incorrect samples was done using SVMs. In the experiments the authors compared the performance of term (base NP) extraction on cleaned and uncleaned data. Depending on the dataset an improvement on the F1-measure of up to more than 40% was achieved, which indicates the importance of text cleaning.

Correcting spelling errors

In Wong et al. (2006) a system is introduced that uses *Integrated scoring for spelling error correction, abbreviation expansion and case restoration in dirty text* (ISSAC) and thus addresses the main sources of noise of online text data. The spelling error correction is based on GNU Aspell[11] by Kevin Atkinson and is used as the base for the error correction. Additionally to the suggestions of Aspell, ISSAC considers possible abbreviations and capitalizations for detected errors. The suggestions are then ranked by a score, which involves edit distance between the candidate and the erroneous word, significance of the candidate and other factors. The system has been tested on a dataset consisting of chat records from 247Customer.com, and the results have been compared to those of using only the suggestions of Aspell. The error correction using only Aspell achieved an accuracy of about 74.39%, which is already a significant improvement. However, including the additional suggestions and the scoring algorithm, an accuracy of 96.56% was achieved.

Thus, if dealing with noisy text corpora, it is worth considering text cleaning as it can improve the final results significantly.

[10] http://www.aclweb.org/aclwiki/index.php?title=NP_Chunking_%28State_of_the_art%29
[11] http://aspell.net/

2.4. NLP Toolkits

To aid the development of NLP systems, a number toolkits is available. On the one hand, there are libraries for special purposes like for instance YamCha[12] (Kudo and Matsumoto, 2001) for text chunking. On the other hand there are general purpose toolkits, which provide a variety of methods for many NLP tasks, some even including a set of text corpora to help developing and testing. A list of tools and related software can for instance be found at the ACL web page[13]. Two toolkits, namely NLTK and GATE, will be introduced in this Section, as they were also used for the implementation in this work.

2.4.1. Natural Language Toolkit (NLTK)

The Natural Language Toolkit[14] (NLTK) provides a collection of python packages which contain methods for tasks reaching from low-level like tokenization or POS-tagging to high-level like classification and clustering. Furthermore it contains a set of corpora, like for instance the Brown Corpus, which is very useful for developing and testing. An advantage of the implementation in python is, that it is platform independent easy to use and comprehensible. Furthermore, the interactive mode of the python console makes it easy to experiment with the tools and quickly learn and understand the basics of NLP.

In the following, an example is provided, that demonstrates the usage of NLTK. For this purpose, an example posting of the dataset which was used in this work will be imported and the basic operations, similar to those performed in the actual implementation, will be performed on it.

Firstly, the posting is saved into a string variable. As the postings come from an online forum, they can contain HTML tags. These can simply be removed using the `nltk.clean_html(...)` method.

```
posting = """<div id="intelliTXT">
   ...:          <br/>No I don't think a placement is likely or at  least not
   ...: one that would be large enough or at a price that would significantly
   ...: hurt HDR's sp in these current circumstances.<br/> <br/>There's no
   ...: need to raise more funds until HDR decides what equity to keep and
   ...: therefore knows its funding requirements.<br/> </div>
   ...:"""

posting_plain = nltk.util.clean_html(posting)
```

Now the next step is to tokenize the sentences. To do that, the Punkt tokenizer (Kiss and Strunk, 2006) is loaded and executed with the plain text as input:

```
sent_tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
sents = sent_tokenizer.tokenize(posting_plain)
```

[12] http://chasen.org/~taku/software/yamcha/
[13] http://www.aclweb.org/aclwiki/index.php?title=Uncategorized_software
[14] http://www.nltk.org/

```
sents
["No I don't think a placement is likely or at least not one that would [...]",
 "There's no need to raise more funds until HDR decides what equity to [...]"]
```

And in the next step, each sentence is tokenized, using the method `nltk.word_tokenize(...)`:

```
sents_tokenized = [nltk.word_tokenize(sent) for sent in sents]

sents_tokenized[0]
['No', 'I', 'do', "n't", 'think', 'a', 'placement', 'is', 'likely', 'or', 'at',
 [...]  'in', 'these', 'current', 'circumstances', '.']
```

With the tokenized sentences, the POS-tagging can now be performed:

```
sents_tagged = [nltk.pos_tag(sent) for sent in sents_tokenized]
sents_tagged[0]
[('No', 'DT'), ('I', 'PRP'), ('do', 'VBP'), ("n't", 'RB'), ('think', 'VB'),
 ('a', 'DT'), ('placement', 'NN'), ('is', 'VBZ'), ('likely', 'JJ'),
 ('or', 'CC'), ('at', 'IN'), ('least', 'JJS'), ('not', 'RB'), ('one', 'CD'),
 ('that', 'WDT'), ('would', 'MD') ('be', 'VB'), ('large', 'JJ'),
 ('enough', 'RB'), ('or', 'CC'), ('at', 'IN'), ('a', 'DT'), ('price', 'NN'),
 ('that', 'WDT'), ('would', 'MD'), ('significantly', 'RB'), ('hurt', 'VB'),
 ('HDR', 'NNP'), ("'s", 'POS'), ('sp', 'NN'), ('in', 'IN'), ('these', 'DT'),
 ('current', 'JJ'), ('circumstances', 'NNS'), ('.', '.')]
```

NLTK doesn't come with a built in NP-chunker, but it provides several possibilities to quickly implement one. For simplicity, in this example a RegEx parser is used. Other possibilities would be, for instance, unigram chunkers or classifier-based chunkers for which NLTK also provides the basic methods and interfaces.

```
grammar = "NP: {<[CDJNP].*>*(<NN[PS]?>|<CD>|<PRP>)}"
parser = nltk.RegexpParser(grammar)
result = parser.parse(sents_tagged[0])
```

Now the result contains the following noun phrases:

```
('NP', [('No', 'DT'), ('I', 'PRP')])
('NP', [('a', 'DT'), ('placement', 'NN')])
('NP', [('one', 'CD')]
('NP', [('a', 'DT'), ('price', 'NN')])
('NP', [('HDR', 'NNP'), ("'s", 'POS'), ('sp', 'NN')])
('NP', [('these', 'DT'), ('current', 'JJ'), ('circumstances', 'NNS')])
```

The first NP is not correct because "No" does actually not belong to it, but the other NPs are correct and also all the NPs contained in the processed sentence were found.

As can be seen, all the steps towards NP-chunking were quite easy and straightforward using NLTK, which demonstrates that the use of such toolkits can be a great help in developing NLP systems.

2.4.2. General Architecture for Text Engineering (GATE)

GATE[15] is an extensive Java based infrastructure that comes in several ways. Thus its components can for instance be integrated in an application using GATE Embedded[16], or it can be used as a standalone tool using the GATE Developer[17] GUI. Either way, GATE provides a wide range of componens like parsers, taggers, information retrieval tools, machine learning plugins and others.

As an example, here it is shown how use GATE Embedded to perform NP-chunking with ANNIE (A Nearly-New Information Extraction System) which is distributed with GATE. The task is performed using a pipeline of ANNIE modules, namely a tokenizer, a sentence splitter, a POS-tagger and a NP-chunker. These can easily be set up in a few lines of code within the corresponding Java module:

```
// load the ANNIE plug-in:
Gate.getCreoleRegister().registerDirectories(
        new File(Gate.getPluginsHome(), "ANNIE").toURI().toURL());
Gate.getCreoleRegister().registerDirectories(
        new File(Gate.getPluginsHome(), "Tagger_NP_Chunking").toURI().toURL());

// create a serial analyzer controller to run ANNIE:
controller = (SerialAnalyserController) Factory.createResource(
            "gate.creole.SerialAnalyserController", Factory.newFeatureMap(),
            Factory.newFeatureMap(), "ANNIE");

//create processing resources and add to controller
//Tokenizer:
ProcessingResource pr = (ProcessingResource)
    Factory.createResource("gate.creole.tokeniser.DefaultTokeniser");
controller.add(pr);
//Sentence Splitter:
pr = (ProcessingResource)
    Factory.createResource("gate.creole.splitter.SentenceSplitter");
controller.add(pr);

[...]
```

The controller can then be linked with a corpus on which the task should be performed. If it is executed, the corpus is annotated corresponding to the executed modules. In this case the corpus is annotated with tokens, sentences, parts of speech and noun phrases and thus the corresponding parts can be extracted from the corpus and used for further processing.

2.5. Summary

As was demonstrated in this Chapter, NLP bears a number of interesting and challenging problems, reaching from low level tasks like segmentation to higher level tasks like information extraction.

[15] http://gate.ac.uk/
[16] http://gate.ac.uk/family/embedded.html
[17] http://gate.ac.uk/family/developer.html

Although research in this area has been done for several decades, computers are still only capable of performing very limited tasks of NLP with satisfying results. This is due to the fact, that natural language doesn't have a well defined structure which describes how information is expressed but in contrast usually provides myriads of ways to express the same thing and is also often ambiguous. Thus, also tasks which seem very simple at the first glimpse, turn out to contain difficulties that are hard to tackle.

Nevertheless, thanks to the effort of a lively community of researchers, many problems are well studied today and some solutions are known and are also applied in practice. Furthermore, the available tools today support the research and development in this area, such that it is easier to concentrate on specific problems instead of being forced to creating all solutions from scratch.

By the introduction into NLP, the first of the three fundamental techniques which are crucial for the implementation was described. Another important technique is OLAP, which is also used in the implementation and is therefore introduced in the next Chapter.

3. Online Analytical Processing

The goal of *Online Analytical Processing* (OLAP) is to support decision makers by providing methods to analyze databases. The basic data structure and operations which can be performed with OLAP are described in this Chapter. Furthermore, an extension to OLAP called Text Cube is described. This extension provides mechanisms to integrate natural language data, in addition to structured data, into the analyzing process.

3.1. Multidimensional Data Analysis with OLAP

The term OLAP was originally introduced in Codd et al. (1993), where the authors investigated multidimensional data analysis and defined 12 rules an OLAP system should maintain. As these rules were considered to be too application specific, in Pendse and Creeth (1995) the authors introduced a more comprehensible set of five rules to state the requirements for an OLAP system:

- **Fast**: The system hast to process the queries fast, within a few seconds at most, to guarantee a steady workflow for the analyst.
- **Analysis**: All relevant analysis operations for the particular application have to be supported. Furthermore, the access to these operations should be provided in a way, that also users, which are not programming experts, can cope with it.
- **Shared**: The system has to implement security mechanisms to provide confidentiality. For multiuser systems this also means, that a system has to handle concurrent read and write access.
- **Multidimensional**: The system has to provide the data in a multidimensional representation. This can be seen as the most important requirement, as it enables the analysis operations which make up OLAP.
- **Information**: All data that is necessary for the analysis has to be provided by the system. Thus, information must not be inaccessible due to technical limitations of the system.

Therefore, to check if a given system satisfies these rules is called the FASMI test (Fast Analysis of Shared Multidimensional Information).

Thus, a central aspect of OLAP is, that the data is organized as a set of points in a multidimensional space, such that the dataset can be represented as a set of points within a hypercube. The datapoints represent the rows of a fact table, e.g. sales records, with *product*, *price*, *quantity*, and

date as dimensions. This data structure with the typical OLAP operations on it is explained in the next Section (3.2). The goal of this structure is, to provide the data in a way, such that it can be viewed from different aspects, and navigated through efficiently.

Given the requirements for OLAP, the multidimensional approach provides several advantages compared to a relational database with SQL as query language. Although in principle, the information that can be queried from a multidimensional database can also be queried using SQL on a relational database, the analyzing process can become a quite tedious task using that approach. This is because the user shall be able to receive the answer to a query in a short time (a few seconds, at most) even if the query is complex. Subsequently, it shall be possible to alter and refine the query and again receive the answer quickly. Furthermore, queries have to be easy to formulate, such that the analyst doesn't have to be a database expert. These requirements can be fulfilled much easier with an multidimensional database than with a relational database.

Analysis Approaches

There are two main approaches of how to analyze the data in an OLAP system, namely the *hypothesis driven approach* and the *discovery driven approach*. A comparison of these approaches and the original definition of the discovery driven approach can be found in Sarawagi et al. (1998). With the hypothesis driven approach, the analyst first makes up a hypothesis, and then analyzes the data to check if the hypothesis holds or not. The goal there is, to find anomalies in the data to discover problems or opportunities for improvements.

In the discovery driven approach, firstly exceptions are precomputed. Exceptions intuitively are values, which remarkably exceed the typical variance of the according dimension values. With this information, the analyst can be supported when navigating through the data cube to find anomalies more efficiently. This can especially be helpful, if the cube has many dimensions. In Giacometti et al. (2009), a method is described, to further improve the discovery driven approach, by recommending queries to the analyst, based on previous investigations.

3.2. OLAP Cube

The *OLAP Cube* is the central data model in OLAP, on which the analyzing operations are performed. As the number of dimensions is arbitrary, the model can actually be seen as a hypercube which contains cells to which the data points belong to. A cell represents a value or a range of values for each dimension of the cube. Thus, each cell contains the set of data points, where the dimension values of the points match the values of the cell. The values, which are then displayed in these cells are called *measures*. The measures can be aggregated with appropriate functions, which are typically sum, or also average, median or other values. The aggregation function is supposed to accurately summarize the underlying data, thus it is important choose an appropriate function. In Horner et al. (2004), for instance, the authors investigated how to handle potentially inaccurate aggregation functions.

An important characteristic of dimensions is, that they can be structured hierarchically. For instance, a geographic dimension could be structured into cities, which belong to states, which

belong to countries, which belong to regions. Thus, the cells in the OLAP cube can represent different levels of such hierarchies of a dimension.

The typical OLAP operations that can be applied on the cube are described in the following subsection.

Operations

There is a number of operations on the OLAP cube, which utilize the hierarchical structure of the dimensions, to efficiently navigate through the data. These operations are:

- **Drill up/down**: These operations concern the hierarchy level of the according dimension. Thus, if drill up is performed, the values of the dimension become more general and therefore the cells in the cube, belonging to this dimension represent a higher level in the hierarchy. For instance, drill up on a geographical dimension, which is currently on level *cities*, will then be on level *states*. Thus, the data points which have city values which belong to the same state values will subsequently belong to the same cell in the cube, according to this dimension. Drill down is the reverse operation of drill up.
- **Slice**: The slice operation restricts a dimension to a certain value of that dimension. E.g. on a geographical dimension with country values, a slice operation could be performed with the value *Austria*. Thus, all datapoints of the resulting cube (or slice) have the value *Austria* in this dimension.
- **Dice**: A repeated application of the slice operation is called dice, as this creates a smaller cube which represents a subset of the original cube.
- **Pivot/Rotate**: As for a cube, with more than two dimensions, not all dimensions can be displayed at the same time. The view of the cube, concerning the displayed dimensions, can be changed with the pivot (aka rotate) operation.

3.3. Types of OLAP

The different types of OLAP concern the way the data is stored. The main types are *Relational OLAP* (ROLAP) and *Multidimensional OLAP* (MOLAP), which are described in the following subsections.

There also exist a number of other types[1] like for instance *Mobile OLAP*, *Web OLAP* or *Desktop OLAP*, which are supported by some implementations. However, these types are not widely supported and do therefore not represent main OLAP techniques.

[1] http://olap.com/w/index.php/Types_of_OLAP_Systems

Multidimensional OLAP and Relational OLAP

In MOLAP, the data is actually stored in a multidimensional database. This has the advantage, that it is in an optimal format for OLAP, and therefore, the OLAP operations can be performed efficiently.

In contrast, ROLAP stores the data in a relational database. This has the advantage, that big databases can be stored more efficiently. However, if the typical OLAP operations are performed on a relational database, the performance is slower than on multidimensional databases, as they are not optimized for such queries.

A survey of these techniques can be found in Pedersen and Jensen (2001). The main aspects where these approaches of data storage differ can be summarized as follows:

- **Query Speed**: As the data structure of MOLAP is optimized for the kind of queries that are performed with OLAP, it outperforms ROLAP concerning query speed. This especially comes into account if a sequence of complex queries has to be computed.
- **Scalability**: Because relational databases are supposed to scale well on very large data sets and are implemented accordingly, ROLAP has an advantage in this aspect. Especially, the computation of dimensions with very high cardinality can become slow in MOLAP systems. Furthermore, a well structured relational database doesn't contain redundancies, while in a multidimensional database, this can not always be guaranteed and therefore the data cube can require a lot of storage space.
- **Precomputation**: In contrast to ROLAP, MOLAP has to precompute the data cube such that views of different hierarchy levels can subsequently be queried efficiently. This also has got to do with scalability, because the precomputation of a large dataset can become quite time intensive.

Therefore, the MOLAP approach is well suited for relatively small datasets, which are queried frequently, while ROLAP scales better on big datasets which are not accessed by a very large number of according queries.

Hybrid OLAP

To achieve a tradeoff between the advantages and disadvantages of these two types, there also exists a hybrid type called *Hybrid OLAP* (HOLAP). With this approach, a subset of the data can be stored in a multidimensional format, which increases query performance on this data, while the remaining data stays in a relational format. Thus a fast query performance on data, which is frequently used, can be achieved, while the size of the cube is kept relatively small.

3.4. Text Cube

The Text Cube was originally introduced in Lin et al. (2008). As modern databases often involve textual data, e.g. the database of an online store might contain user comments about products, it makes sense to try to utilize this data in an analysis process. For this purpose, the Text Cube uses term hierarchies as additional dimensions to an OLAP cube. Furthermore, information retrieval (IR) measures, namely term frequency and inverted index, are supported by the Text Cube.

While IR measures are relatively easy to implement, the term hierarchies imply, that such hierarchies have to be created in advance. As different datasets might contain very different terms, which are relevant, especially if the datasets are from different domains, specialized term hierarchies are required for each dataset. Thus, in the implementation in this work, an automatic taxonomy extraction was applied on the dataset in advance, to find such hierarchies and subsequently include them in the OLAP system to utilize the additional capabilities of the Text Cube.

Text Cube Operations

With the extracted text data and the term hierarchies, the following additional operations can be defined:

- **Pull-up**: This operation can be performed on dimensions that represent term hierarchies, and is similar to the drill up operation. The reason for the need of a specialized operation on the term hierarchies is, that the leaf nodes are not necessarily all at the same level, thus the levels have to be defined differently. Therefore, only the lowest and the highest levels are defined initially. If pull-up is performed on a given level, other than the root level, with a certain node (term) of that level, the parent of that node is added to the new level, and all its descent nodes are removed. Thus, the result is a higher level.
- **Push-down**: Push-down is the reverse operation of pull-up. Thus, this operation yields a lower level, by removing a node, which has child nodes, and adding its children to the new level.
- **Keyword search**: This operation restricts the datapoints to those, that contain the specified keywords in an according textual dimension. Thus, the resulting cube contains a subset of the datapoints of the original cube.
- **Term frequency**: For a given cube, term frequency yields a list of terms, which occur in the according textual dimension of the datapoints, for each cell of the cube. The lists are ordered by the number of occurrences of the corresponding terms. Thus, for instance, the n most frequent terms can be queried for each cell.

With these operations, an analyst has additional possibilities to investigate a given dataset, as additional information can be utilized.

3.5. Summary

OLAP provides a powerful tool to analyze multidimensional databases. This analysis can help decision makers to discover problems and new opportunities by investigating business relevant data that is contained in the database of the according enterprise. To aid this analyzing process, the OLAP cube provides efficient methods to navigate through big datasets.

The different types of OLAP, which define how the data is stored, allows to adjust an OLAP system to the needs of the application. Additionally to the traditional OLAP approach, which is only capable of handling structured data, Text Cube extends the OLAP model to make use of textual data. Thus, additional data can be utilized which provides further possibilities to analyze according databases.

The methods which were described in this Chapter have also been implemented in the prototype. The database, which provides the data for the OLAP system is implemented such that it makes use of semantic web technologies, which are described in the next Chapter.

4. Semantic Web Technologies

In this Chapter, an introduction to the basic techniques on which the *Semantic Web* is built upon is given. Most of these techniques have been standardized by the W3C in about the last decade and have been taken up by researchers and industry to implement tools and products that make use of these techniques. Such tools have also been used in the implementation to develop an RDF data store, which is described in Chapter 6.

4.1. Semantic Web

The concept of a semantic web was originally introduced by Tim Berners-Lee (Berners-Lee et al., 2001). The goal is, to add structured data to the web, which can be read and processed by computers. The idea is to make the data available like in a database, such that it is possible to answer questions like "*Who was the first person who climbed all mountains which are higher than 8000m?*" or to enable software agents to automatically perform tasks like "*Make an appointment at a nearby dentist in the next week.*", where the agent would additionally have to take care of other tasks already in the calendar.

Such things can not easily be done if the information in the web comes as natural language or media files only, as the structure that comes with hypertext, only enables browsers to display web pages in a way which is suitable for humans. But this does not enable the computer to extract information from the data. To address this issue, the *Resource Description Framework* (RDF) was introduced to provide a format that can represent structured data. How this is done will be explained in the next Section (4.2).

Furthermore, it is important that providers of information do agree on a common set of vocabulary such that datasets which represent the same type of information are always recognized as such. For that purpose, ontologies are used, which define relations between terms and map the data to the according semantics. This is also described in more detail in the following sections.

4.2. Resource Description Framework (RDF)

RDF is one of the core technologies of the semantic web and was originally introduced and specified by the W3C[1] (Carroll and Klyne, 2004). It provides the basic concepts to represent data in

[1] http://www.w3.org/

a general way such that arbitrary entities with their according attributes and relations can be represented. In the following subsections the data structure of RDF and according serialization formats will be described in more detail.

4.2.1. RDF Structure (Graph Model)

In RDF each entity (resource) is identified by a URI and its properties are defined by triples of the form *(subject, predicate, object)*, where the *predicate* defines the property which maps the value of *object* to the *subject*. Each of the elements in the triple can be a URI, where the object can also be a literal which are used to identify values like numbers or dates. Thus, an entity can be associated with any number of properties or related to other entities by defining the according set of triples. The result can then be seen as a directed graph, where the nodes represent entities and values (*subject* and *object*) and the edges represent the relations (*predicate*). An example is given in Figure 4.1.

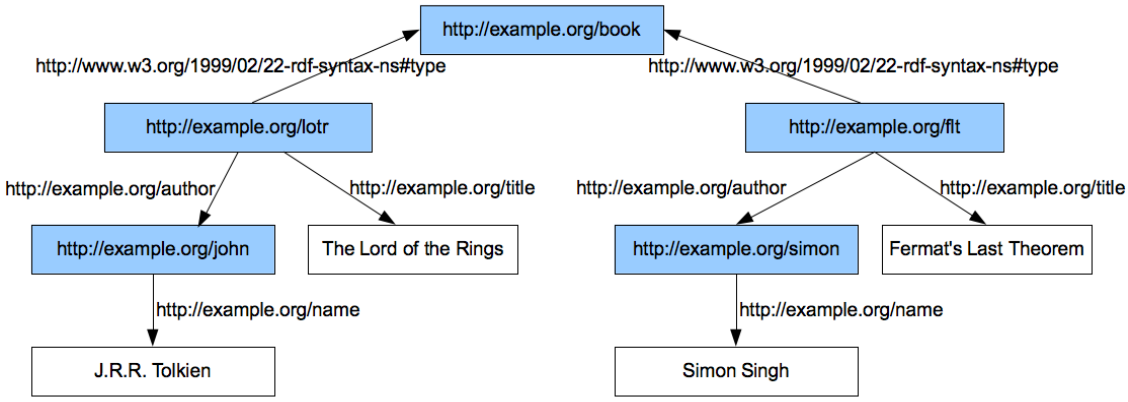

Figure 4.1.: Here the colored nodes represent resources and the white nodes represent literals. For instance the resource `http://example.org/lotr` is associated to another resource `http://example.org/john` via the property identified by `http://example.org/author`. This relationship can also be seen as a triple consisting of *subject*=`http://example.org/lotr`, *predicate*=`http://example.org/author` and *object*=`http://example.org/john`.

Literals and Datatypes

As mentioned before, the object of a triple can not only be a URI but also a literal. A literal can either be plain, thus being a string with an optional language tag, or it can be a typed literal which is a string with an additional datatype URI. Datatypes are important for programs that are intended to automatically process the data as they need to know how to interpret a given literal. As RDF itself only comes with one special built-in datatype, namely the *XMLLiteral*, which is defined as the set of all valid XML strings, all other datatypes have to be specified externally. The intention is, that common datatypes, like those defined by the XML Schema (Malhotra and Biron, 2004) should be used.

Furthermore, if datatypes are needed that are not covered by any existing definitions they can be defined, where the definition must contain the lexical space, the value space and a lexical-to-value

mapping:

- **lexical space**: The set of strings that correspond to the values of the datatype.
- **value space**: The set of values which the datatype can take.
- **lexical-to-value mapping**: A mapping from the elements in the lexical space to the corresponding elements in the value space such that each lexical value maps to exactly one element in the values space.

Blank Nodes

Blank nodes are a special type of nodes which do not have a URI. Thus, these nodes are not accessible from outside the database they belong to, but must nevertheless be uniquely identifiable inside the database. The purpose of such nodes is illustrated in Figure 4.2.

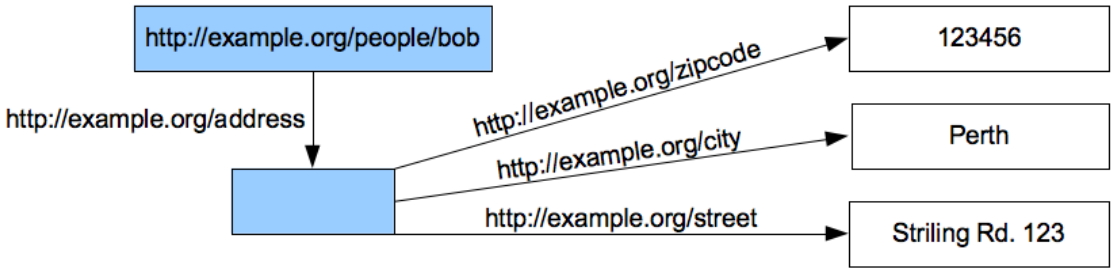

Figure 4.2.: Here the resource `http://example.org/people/bob` is associated with an address via a blank node. Such blank nodes can be introduced if multiple objects (here: zip-code, city and street) which belong to a common concept (address) have to be associated with one subject. If this intermediate node is not intended to be referred to directly from outside, it doesn't need a URI, thus it can be created as a blank node.

4.2.2. Serialization Formats

The originally specified format to serialize the abstract model of an RDF graph is RDF/XML (Beckett, 2004). This is an XML based format that allows to represent the RDF graph in a machine readable way.

The root element of an RDF/XML document usually is `<rdf:RDF>` which can contain namespace definitions (this element can also be omitted if it would contain only one child). For instance, for the graph of Figure 4.1 this could be:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/">
...
```

with the two defined namespaces for `rdf` and `ex`. The properties of the resources are described with the `<rdf:Description>` element, where the according resource is defined with its URI in the `about` attribute. The according properties can then be embedded as children of that element or

for string literals also as attributes. For instance, a subset of the nodes and edges of the example in Figure 4.1 could be serialized as:

```
1  <rdf:Description rdf:about="http://example.org/lotr"
2                   ex:title="The Lord of the Rings">
3      <rdf:type rdf:resource="http://example.org/book" />
4      <ex:author>
5          <rdf:Description rdf:about="http://example.org/john" />
6      </ex:author>
7  </rdf:Description>
8
9  <rdf:Description rdf:about="http://example.org/john"
10                  ex:name="J.R.R. Tolkien" />
```

In this example, the attribute `ex:name` of the description about `http://example.org/john` in line 10 could also have been placed inside the description tag in line 5. Thus the way an RDF graph is serialized is not unique.

Typed literals can be associated with their type via the `rdf:datatype` attribute, for instance:

```
<rdf:Description rdf:about="http://example.org/john">
    <ex:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
     >J.R.R. Tolkien</ex:size>
</rdf:Description>
```

Here the literal "J.R.R. Tolkien" is associated with the datatype string.

Furthermore if a resource is of a certain type, like in the example `http://example.org/lotr` is of the type `http://example.org/book`, then the description tag for the resource can also be replaced by a tag with the according type, for instance:

```
<ex:book rdf:about="http://example.org/lotr">
    <ex:title>The Lord of the Rings</ex:title>
</ex:book>
```

By defining the additional namespace `xml:base="http://example.org/"` the attribute `rdf:about="http://example.org/id"` can also be abbreviated to `rdf:ID="id"`.

These are the basic concepts to serialize an RDF graph with RDF/XML (the complete definition can be found in Beckett, 2004). Another way to serialize a graph is with *Notation 3* (N3) as described in the following.

Notation 3 (N3)

N3[2] was introduced to provide a more convenient and readable alternative to RDF/XML. Documents written in N3 usually start with namespace declarations. These are defined by the keyword `@prefix` followed by a identifier and a colon and the URI of the namespace within angular brackets. This statement, like all other statements in N3, is terminated by a "." and also URIs are always written in angular brackets. For instance, for the example in Figure 4.1 this could be:

[2] http://www.w3.org/TeamSubmission/n3/

```
@prefix ex: <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <#> .
```

where the last prefix defines the current document, thus an element of the current document can be referred to by a colon followed by the name of the element. Other elements within these namespaces can be referred to by the specified name followed by a colon and the name of the element. E.g. `rdf:type` would refer to the URI
`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

Triples can then be defined by statements of the form "`subject predicate object.`" where each of subject, predicate and object can be a URI (or an abbreviation using the names of the defined namespaces). The object can also be a literal which is written in double quotes. For instance, a few statements of the graph in Figure 4.1 would be:

```
ex:lotr rdf:type ex:book .
ex:lotr ex:author ex:john .
ex:lotr ex:title "The Lord of the Rings" .
```

If several statements are made belonging to the same subject, this can be abbreviated by writting the subject only once and separating the predicate/object pairs by semicolons, e.g.

```
ex:flt rdf:type ex:book ; ex:author ex:simon ;
    ex:title "Fermat's last Theorem" .
```

Thus N3 provides a way to serialize RDF graphs, which is more convenient to read and write manually, compared to RDF/XML.

4.2.3. RDF and RDF Schema Vocabulary

The vocabulary of RDF is (intendedly) quite limited. Therefore, *RDF Schema* (RDFS) defines a further set of vocabulary to describe and structure resources (Guha and Brickley, 2004).

Classes and Properties

In RDF everything is a resource, thus an instance of `rdfs:Resource`, which is the class of everything and is a superclass of all other classes. Classes are used to group objects together and are recursively defined such that `rdfs:Class` is an instance of `rdfs:Class` (and also `rdfs:Resource` is an instance of `rdfs:Class`). Other classes defined by RDFS are `rdfs:Literal` which is the class of all literals and `rdfs:Datatype` which is the class of all datatypes (literals and datatypes as described in Section 4.2.1 (Literals and Datatypes).

Another important class of RDF is `rdf:Property` as a set of other classes is derived from it in RDFS. Properties describe the relations between subjects and objects, thus they are the predicates in the triples. The subclasses of `rdf:Property` in RDF(S) are:

- `rdf:type` - defines that a subject is an instance of the class defined by the object.
- `rdfs:domain` - a triple `P rdfs:domain C` specifies the domain for property `P`. Thus for a triple `S P O`, the subject `S` has to be of class `C`.
- `rdfs:range` - a triple `P rdfs:range C` specifies the range for property `P`. Thus for a triple `S P O`, the object `O` has to be of class `C`.
- `rdfs:subClassOf` - defines that one class is a subclass of another class. Thus hierarchies of classes can be created.
- `rdfs:subPropertyOf` - defines that one property is a subproperty of another property. Thus if `P1 rdfs:subPropertyOf P2` and `A P1 B` then also (implicitly) `A P2 B`.
- `rdfs:label` - defines a label for a resource in a human readable way.
- `rdfs:comment` - defines a human readable comment (description) for a resource.

Containers

RDF(S) also describes some basic container classes with the according properties to define which resources belong to a container. These are subclasses of `rdfs:Container`. The membership of a resource to a container is defined by the property `rdfs:member`. The different subclasses of `rdfs:Container` provide mechanisms to define ordered and unordered containers.

Thus with the extended set of vocabulary, RDF gains additional expressiveness to describe resources. The full set of RDF(S) vocabulary can be found in the according W3C recommendation in Guha and Brickley (2004). In Section 4.3 the *Web Ontology Language* is described which is built upon RDFS and provides further expressiveness to describe resources and their relations.

4.2.4. Query Languages

Query languages provide the possibility to query data from an RDF graph, similar to query languages on relational databases. For RDF, a number of query languages exist which are supported by a variety of implementations in different programming languages (a survey of different query languages can be found at the W3C website[3]).

The query language that was defined by the W3C is called SPARQL (*SPARQL Protocol and RDF Query Language*) which is an official W3C recommendation (Prud'hommeaux and Seaborne, 2008).

4.2.4.1. SPARQL Queries

The syntax of SPARQL queries is similar to that of N3. The first part of a query is usually a set of definitions of prefixes in the form of `PREFIX name: <URI>` to define the used vocabularies. The next thing is the keyword that defines the type of the query, like `SELECT` (the other types

[3] http://www.w3.org/2001/11/13-RDF-Query-Rules/

of queries are described later). The keyword `SELECT` is followed by a set of variables, which are marked by an initial "?". This is followed by the keyword `WHERE` followed by a set of statements between curly brackets. The statements are triples where each of subject, predicate and object can be a variable. For instance, a query on the graph of the example in Figure 4.1 that should return all resources of type `ex:book` would be:

```
PREFIX ex: <http://example.org/>
SELECT ?b
WHERE { ?b a ex:book . }
```

where `a` in the triple is a short form for `rdf:type`. The resultset would then consist of all possible bindings of the variable `?b` which are in the example `ex:lotr` and `ex:flt`.

Literal values can be queried by putting the according value between double quotes and appending the type by `^^<DATATYPE_URI>`. E.g. to query all books with author "Simon Singh" (of type `xsd:string`) this would be:

```
SELECT ?b
WHERE {
    ?b ex:author ?a .
    ?a ex:name "Simon Singh"^^<http://www.w3.org/2001/XMLSchema#string> . }
```

which would return a result set with the one element `ex:flt`.

To restrict the values of a variable the keyword `FILTER` can be applied. The condition of `FILTER` depends on the datatype of the according variable, e.g. if the string variable `?name` should be filtered using a regular expression such that only values that start with "Simon" should be considered, this can be done by `FILTER regex(?name, "^Simon")`.

Triples in the where clause can also be made optional by using the keyword `OPTIONAL` thus variables can be unbound in the result set. To match alternative sets of triples the keyword `UNION` can be used, e.g. if all bindings for a variable `?x` for alternative properties `P1` and `P2` with value "value" should be found this can be done by:

```
SELECT ?x
WHERE { {?x P1 "value".} UNION {?x P2 "value".} }
```

The result set can also be ordered according to a variable using the keyword `ORDER`. Furthermore it can be limited with the keyword `LIMIT` and an offset can be applied using the keyword `OFFSET`. If multiple solutions according to a variable should be suppressed, the keyword `DISTINCT` can be put in front of that variable in the select clause.

Other types of Queries

In addition to the select query, which returns a set of bindings for the queried variables, three other types of queries exist. The difference between the types lies in the results they produce:

- **Construct Queries**: The result of a construct query is an RDF graph. Thus a graph pattern is given in the construct clause such that each proper binding of the pattern is in the resulting graph.

- **Ask Queries**: Ask queries return a boolean value. If the specified pattern of triples has a solution then `true` is returned, else `false`.
- **Describe Queries**: The result of a describe query is also an RDF graph. Therefore, for all valid bindings of the given variables all according descriptions (`<rdf:Description>` about the according resource) are returned.

4.3. Web Ontology Language (OWL)

The *Web Ontology Language* (OWL) is also a standard defined by the W3C (McGuinness and van Harmelen, 2004). It is built upon RDF(S) and its purpose is to add further vocabulary with according semantics to gain additional expressiveness. The basic components are described in Section 4.3.1. There are three sublanguages of OWL, namely OWL Lite, OWL DL and OWL Full. The differences between the sublanguages are in the expressiveness, decidability and completeness, which is described in more detail in Section 4.3.2. The current version is OWL 2 (Krötzsch et al., 2009), which is based on and compatible with the previous versions, but adds further expressiveness.

4.3.1. Components and Semantics

The vocabulary added by OWL mainly concerns three components, namely classes, properties and individuals. If resources are described with this vocabulary, it is possible to automatically reason over a created ontology such that implicit knowledge can be derived. Implementations of reasoners are introduced in Section 4.4.

Classes

Classes are, like in RDFS, collections of things that share some properties. The set of resources that belong to a class is called the extension of that class. There are six possibilities to define a class:

1. **Class Identifier**: simply defines a class by giving it a name, e.g. `<owl:Class rdf:ID="Book" />` defines a class named `Book`.
2. **Enumeration**: enumerating the individuals that belong to the class. This class then contains exactly the individuals of that list. E.g. (note: all individuals in OWL are instances of `owl:Thing`):

```
<owl:Class rdf:ID="Gender">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="http://example.org/Female" />
    <owl:Thing rdf:about="http://example.org/Male" />
  </owl:oneOf>
</owl:Class>
```

where the attribute `rdf:parseType=''Collection''` defines that the children of the according element form a list.

3. **Property restriction**: defines an anonymous class by restricting the values or cardinality of a property. E.g. a class with value restrictions:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="http://example.org/hasGenre" />
  <owl:someValuesFrom rdf:resource="http://example.org/Fantasy" />
</owl:Restriction>
```

   defines a class of individuals that have genre *Fantasy*.

4. **Intersection, Union and Complement**: defines a new class by performing the according set theoretic operations on the given class(es). E.g.

```
<owl:Class rdf:ID="EdiblePlant">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="Plant" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://example.org/isEdible" />
      <owl:hasValue rdf:dataType="&xsd;boolean">true</owl:hasValue>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

   defines the class *EdiblePlant* by the intersection of the class *Plant* and the class of all things that are edible.

Furthermore, three additional statements about classes are possible. The first one, namely `rdfs:subClassOf`, has already been described in the according section about RDFS. The others are:

- **equivalentClass**: this states that two classes share exactly the same extension. This does not mean, that the concepts described by the classes must be equal or that they must have the same properties, but only that they contain the same individuals.
- **disjointWith**: this states that the two sets of individuals which belong to the two classes (the class extensions) are disjoint.

Properties

There are two main types of properties in OWL, namely `owl:ObjectProperty`, where the range is a set of individuals, and `owl:DatatypeProperty`, where the range is a datatype. Both are subclasses of `rdf:Property`, thus all the constructs of RDFS like `rdfs:domain` or `rdfs:subPropertyOf` also apply to OWL properties.

Additionally the following statements can be made about properties:

- **owl:equivalentProperty**: states that two properties have the same subject/object pairs associated with them.

- **owl:inverseOf**: states that two properties have the same subject/object pairs associated with them, but with subject and object swapped. Thus the properties in this case have to be of type `owl:ObjectProperty`.
- **owl:TransitiveProperty**: if a property is of that type, then for subject/object pairs (a, b) and (b, c) with that property, the property also holds for (a, c).
- **owl:SymmetricProperty**: if a property is of that type, and a subject object pair (a, b) has that property, then also (b, a) has that property.
- **owl:FunctionalProperty**: this type of property restricts the cardinality such that for subject/object pair (a, b) with this property, the pair (a, c) cannot also have this property.
- **owl:InverseFunctionalProperty**: this is the inverse of a functional property. Thus if an object b is associated to a subject a via a inverse functional property, b cannot also be associated to a subject c via the same property.

Individuals

Individuals are instances of classes and are referred to by an URI. Generally, different URIs do not necessarily refer to different concepts, thus with OWL, several statements about the identity of individuals can be made.

To state that two URIs actually represent the same concept, the property `owl:sameAs` can be used. This is for instance useful, if two ontologies shall be mapped to each other. In contrast, it can also be stated, that two URIs represent different concepts using the property `owl:differentFrom`. For a set of URIs where all represented concepts are pairwise disjoint, OWL provides the class `owl:AllDifferent` which can be instantiated with a list of individuals which are all pairwise disjoint, e.g.:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Book rdf:about="http://example.org/FermatsLastTheorem" />
    <Book rdf:about="http://example.org/LordOfTheRings" />
    <Book rdf:about="http://example.org/ComputersAndIntractability" />
  </owl:distinctMembers>
</owl:AllDifferent>
```

4.3.2. Sublanguages

As mentioned earlier, OWL defines three sublanguages, where OWL Full contains the full set of OWL concepts without restrictions, and is thus the sublanguage with the greatest expressiveness. Furthermore, each proper RDF document is also a valid OWL Full document, which is not true for OWL DL and OWL Lite. However, the expressiveness of OWL Full comes with the drawback, that neither is a computation on the data guaranteed to finish in finite time, nor can it be guaranteed, that all valid conclusions can be drawn (completeness and decidability).

Therefore, OWL DL (the name is derived from description logic) sacrifices some expressiveness to gain completeness and decidability. The vocabulary of OWL DL is the same as that of OWL Full, but comes with some constraints on the usage of some concepts and also most of the RDF(S) vocabulary is restricted in OWL DL. Thus, for instance, an individual cannot be a class at the same time. Therefore, OWL DL is appropriate if completeness and decidability is required while the full expressiveness is not necessary.

For OWL Lite, the constraints of OWL DL also apply, and furthermore a subset of the OWL vocabulary is restricted. OWL Lite still provides the possibility to create class hierarchies and simple property definitions and is considered as the minimal useful subset of language features[4].

4.3.3. Existing Ontologies

To bring the semantic web to its full potential, it is necessary to widely agree on a common set of vocabulary. For this purpose, a number of ontologies (or vocabularies) has been developed, which are used in many different systems. These reach from general ontologies to very specialized domain ontologies.

General Ontologies - DC and FOAF

An early vocabulary was the so called *Dublin Core*[5] (DC). This was originally developed in 1995, thus even before RDF was developed. To date it is internationally standardized (by ISO[6], IETF[7] and ANSI/NISO[8]), used in many applications and also available in RDF format. It defines 15 general terms to describe a resource, which can be for instance a document, website, video file or physical object. The vocabulary provides terms to describe the content (e.g. `dc:title`, `dc:description`), creators (e.g. `dc:creator`, `dc:contributor`) and other metadata about the resource. Another wide spread ontology is FOAF[9] (*Friend-Of-A-Friend*) which also makes use of DC (among others). The main purpose of this ontology is to relate people and their activities of different social networks. Some basic terms of the vocabulary are for instance the class `foaf:Person`, of which the instances describe real (or imaginary) persons or the property `foaf:homepage`, which relates something (e.g. a person or a project) with its homepage.

FOAF is also used in many applications and also investigated in several research papers. For instance, in Golbeck and Rothstein (2008) the authors investigated the intersection of FOAF data in different social networks, and found out that a significant amount of data from the different networks could be merged, which demonstrates a potential application of semantic web data. A very different application which makes use of FOAF is introduced in Banford et al. (2010). This application makes use of smartphones with Bluetooth to detect co-present users of a network. If users are detected, which haven't yet been introduced to each other but have friends in common, the system can notify the users about each other.

[4] http://www.w3.org/TR/owl-ref/#OWLLite
[5] http://dublincore.org/documents/dces/
[6] http://www.iso.org/iso/search.htm?qt=15836&searchSubmit=Search&sort=rel&type=simple&published=on
[7] http://www.ietf.org/rfc/rfc5013.txt
[8] http://www.niso.org/standards/z39-85-2007/
[9] http://www.foaf-project.org/

Domain Ontologies

There also exist a number of domain-specific ontologies like for biology, geography or medicine which are used by specific tools. The *Cell Cycle Ontoloy*[10] (CCO) for instance, is used to describe the cell division process and involves a variety of other related ontologies. BioPAX[11] is a language for biological pathway data, which also uses an OWL based format to exchange data. Thus it provides mechanisms to exchange, aggregate and visualize data from different databases.

Therefore, the potential utility of ontologies and the techniques to realize them is not limited to any certain domain, but can be applied in many fields and for a variety of different purposes.

4.4. Semantic Web Tools and Programming Libraries

To date, a variety of tools is available to aid the development and usage of semantic web related techniques. On the W3C RDF website[12] for instance, a list of currently more than 170 RDF relevant tools is available. Among these are both commercial and non commercial tools which support all common platforms and programming languages. The purposes of the tools reach from triple stores with according implementations of query languages, to reasoners and ontology browsers.

Two such tools were used in the development of the prototype in this work, namely Sesame and Protégé. Sesame[13] provides an open source Java implementation of an RDF triple store. It provides several ways to set up and access repositories including according query language support. Furthermore, its functionality can be extended with plugins, for instance to integrate reasoners or to link it with other tools.

Protégé[14] is a Java based open source tool for ontology construction. It provides a GUI for entering classes, properties and individuals, mechanisms to import and export ontologies into various formats and for other related tasks. Additionally it can be extended with plugins, which for instance provide graphical visualizations for ontologies or to integrate reasoners. There also exists a plugin to access Sesame repositories from Protégé.

Reasoners

As the definition of OWL is related to logic, it is possible to perform reasoning tasks on ontologies. Therefore, a number of implementations of reasoners are available like for instance FaCT++[15] or Pellet[16], which are open source implementations in C++ and Java respectively, or RacerPro[17],

[10] http://www.semantic-systems-biology.org/cco/
[11] http://www.biopax.org/
[12] http://www.w3.org/RDF/
[13] http://www.openrdf.org/
[14] http://protege.stanford.edu/
[15] http://owl.man.ac.uk/factplusplus/
[16] http://clarkparsia.com/pellet
[17] http://www.racer-systems.com/products/racerpro/index.phtml

which is a commercial reasoner (but free for research and teaching). Such implementations are usually focused on OWL DL, as it is complete and decidable, thus all true statements can be inferred, and all computations are guaranteed to finish in finite time. Furthermore, the implementation of a reasoner is not a trivial task, thus different implementations of reasoners follow different design goals like scalability, efficiency or completeness. In Bock et al. (2008) for instance, the authors benchmarked different reasoners with main focus on query response time and found out, that the reasoners have according strengths and weaknesses. Thus, if a reasoner is used in an application, it makes sense to investigate, which reasoner best fulfills the requirements.

In contrast to query languages like SPARQL, which are only capable of finding explicitly stated knowledge, reasoners can also derive implicit knowledge, by applying logical rules of inference. Thus typical tasks for reasoners are for instance to check the consistency of ontologies and classes or deriving implicitly stated superclasses of a class.

4.5. Summary

The semantic web technologies provide the basis for the shift from a web, where the content is interpretable only by humans, to a web where also computers can "understand" the available information. For that purpose, the framework for knowledge representation RDF and the web ontology language OWL, which allows to define the semantics of the stored data, have been specified. If these techniques are utilized, it is possible to automatically reason over the given knowledge, to combine knowledge bases or to perform other tasks which are not possible with the common knowledge representation in the web. This does however not mean, that the semantic web technologies are intended to replace the current techniques, but to replenish them.

The semantic web technologies have already been applied in a number of projects like for instance DBpedia[18], FOAF and many more and also a variety of tools has been implemented which support these technologies.

In the implementation RDF and OWL are used to set up a datastore, which is one of the three main modules of the prototype. As now the three fundamental techniques that are used in the implementation have been introduced, the requirements for the implemented system and the conceptual approach is described in the next Chapter, followed by a detailed description of the implementation in Chapter 6.

[18] http://dbpedia.org/

5. Requirements and Approach

The techniques that were discussed in the previous chapters shall now be brought together to set up a new system that combines their strengths. As the Text Cube can make use of taxonomic dimensions but does not define how to achieve them, an automatic taxonomy extraction shall be used for that purpose. Furthermore, semantic web technologies can be utilized to provide a flexible datastore, that can easily be extended by additional information at any time.

Thus the goal is to implement a system that provides OLAP and Text Cube functionality in a way that extracted information can easily be integrated. Such information has to be provided by automatic taxonomy extraction that is performed on the available textual data. The database has to be stored in a format such that it is extensible and provides an uncomplicated possibility to add new information to an existing database. The requirements are defined by the following points:

- **Datastore**: The datastore has to provide the possibility to extend an existing database with additional information at any time. The method to store and access the data should also be extensible in a way, that data from several sources can be aggregated. Methods to import and export data, especially data that can be used by OLAP, have to be provided. Thus, a simple way has to be provided, to identify the data, that is relevant for OLAP, while the original data must not be altered.
- **Natural Language Processing**: NLP has to be performed, such that the methods defined by Text Cube can be utilized. Therefore, the relevant data from a specific database has to be queried and the following information has to be extracted:
  - **Term frequency**: The term frequency for the documents in the database has to be extracted, to be able to run information retrieval queries on the documents which are stored in the database.
  - **Taxonomies**: An automatic taxonomy extraction has to be performed to extract additional information from NL data that can be utilized by the OLAP system.
- **OLAP and Text Cube**: The functionality defined by OLAP and Text Cube has to be supported. Thus the data that is stored in the datastore, including the data that has been extracted by NLP, has to be imported and brought into an appropriate format to perform the OLAP and Text Cube operations.

5.1. Conceptual Approach

The implementation consists of three main parts, namely a datastore, a natural language processor and an OLAP engine. This Section gives an overview, of how the techniques, described in the previous chapters, are brought in to realize the system.

5.1.1. Datastore using Semantic Web Technologies

To be able to access databases in a general and convenient way, an RDF representation was used. The main reason for that is, that RDF allows to create a general representation for the OLAP dimensions, using a simple ontology, as described in 6.3.2. This enables the possibility to map any database, which is available in RDF format to the ontology, and thus being able to query the data with predefined SPARQL queries. Therefore, the database itself doesn't need to be altered but can be imported into the OLAP system using these queries. Furthermore, the other modules of the system don't need to know the structure of a specific database, but can query the needed data by utilizing the mapping and the predefined queries.

The RDF representation also makes the framework extensible. In the prototype, a separate RDF repository is used, which is however not the only way a database could be accessed. Some data warehousing systems, like for instance Virtuoso[1] provide the possibility to aggregate data of several different sources and represent it as a single database in the desired format. Thus, for instance, if some data is available partly in an SQL database and partly as RDF data, the whole dataset can be aggregated and be mapped to a single RDF representation, which can in turn be mapped to the OLAP ontology. Similarly, if the data was stored in a relational database and mapped to an RDF representation via the D2R Server[2], this representation could again easily be mapped to the OLAP ontology.

5.1.2. Natural Language Processing

Natural language processing is used to extract two types of information, firstly term frequency of the documents to be able to run information retrieval queries, and secondly taxonomy extraction.

Term Extraction

Term extraction is a fundamental task in NLP and therefore several NLP frameworks can be used to perform it as they provide the according methods. Thus a module was implemented that makes use of such methods. The steps this module performs are to query the textual data from the RDF repository, to perform the term extraction and to write back the extracted data.

[1] http://virtuoso.openlinksw.com/
[2] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

Taxonomy extraction

The first step for the taxonomy extraction is to perform NP-chunking on the given documents. The resulting set of NPs, which represents a set of concepts, is then a starting point for the relation finding. The reason why NPs are used instead of simple terms is that NPs are more likely to represent more specific concepts as they are a composition of a number of simple terms.

The reason why only taxonomies are extracted instead of more general ontologies is that the result should be fed into the OLAP system, and therefore only hierarchical structures can be utilized.

The relations are extracted by querying the Wikipedia categorization system, based on the idea in Wong (2008). This has several reasons. First of all, Wikipedia is a huge source of background knowledge which also contains very new knowledge, due to the collaborative effort of the big community that maintains it. Furthermore, the relations are correct with high probability as they are entered and checked repeatedly by humans. Also the variety of topics of many different fields is important for this approach, because the system should be able to extract taxonomies from any domain, even without knowing the domain of the processed corpus in advance. The reason for that is, that for input data like the *Australian Stock Forum Data*, that was used in the experiments, the documents can be of a whole variety of domains. For instance, in the used dataset discussions about medicine companies might contain very different domain specific terms than discussions about mining companies. This is also an additional challenge in finding domain specific terms.

The last step of the taxonomy extraction, which is performed to increase the quality of the taxonomies, is pruning. Pruning is a technique, that is used on tree-like structures, which serves several purposes, depending on the application. For instance, in Recio-Garcia and Wiratunga (2010), taxonomy pruning is performed for disambiguation. Furthermore, it is also commonly performed on decision trees to prevent overfitting (e.g. Witten and Frank, 2005; Han and Kamber, 2006). Here the pruning steps deal with nodes, that do probably not fit to the according taxonomy and shall therefore be removed.

5.1.3. OLAP Engine

The OLAP engine is capable of importing the data, that is queried from the repository, and performing several analyzing operations on it. Among these methods are typical OLAP methods like slice or drill up/down. Furthermore, it is capable of performing the Text Cube methods pull-up, push-down on extracted taxonomies and information retrieval queries on extracted terms.

As the OLAP engine is a separate module, any data that is in the database or has previously been extracted can be passed on to it. Thus, additionally to the conventional analyzing approach of OLAP, a document driven analyzing approach is possible, where the datapoints represent documents with their extracted features as dimension values.

5.2. Related Work

Based on the idea of Text Cube and related ideas several research projects have been developed in recent years. In Zhang et al. (2009) for instance, a model called *topic cube* is proposed which integrates a topic hierarchy into OLAP as an additional text dimension. This idea combined with others was also further developed in Yu et al. (2009). There a model called *iNextCube* is proposed, which further extends the data cube model.

Thus, there is also the possibility to integrate other types of information that can be extracted from natural language text into the Text Cube. This could be for instance the information extracted by sentiment analysis (e.g. Pang and Lee, 2008; O'Hare et al., 2009; Choi et al., 2009), document classification (e.g. Ko and Seo, 2009; Salles et al., 2010; Song et al., 2006) or named entity recognition (e.g. Kozareva, 2006; Whitelaw et al., 2008).

Another cube model which is used for information retrieval is proposed in Janet and Reddy (2010) and Janet and Reddy (2011), where the index to retrieve the documents is stored in a cube.

The ontology/taxonomy extraction in this work is mainly based on the ideas in Wong (2009). However, there are also many other research groups that work on that topic which follow different approaches. Big differences in these approaches lie in their type, semi-automatic (e.g. Carvalheira and Gomi, 2007) or automatic (e.g. Guo, 2007) and how they extract terms and relate terms, where for instance Sanchez and Moreno (2008) and Rosenfeld and Feldman (2007) follow very different approaches, both with their strengths and weaknesses.

Also several related issues of data storage or data warehousing are addressed in a number of research papers. E.g. Merritt (2002) and Boussaid et al. (2008) deal with data warehousing using web technologies or with data from the web. In Ding (2007) issues about provenance and searching in RDF data are discussed. There are also several commercial products, like for instance Virtuoso, which provide RDF stores and also possibilities to aggregate data from several sources.

6. Implementation

In this Chapter, the implementation of the prototype is presented. First a short overview of the system architecture is given in Section 6.1, followed by a more detailed description of the implemented components in the following sections. Finally, the results, and the capabilities of the prototype are demonstrated in Section 6.5.

6.1. System Overview

The implemented system consists of three main parts which are responsible for data storage, natural language processing and OLAP respectively. The interplay of these modules can be seen in Figure 6.1.

The *Data Store* contains the RDF repository, which is implemented using the Sesame library, which comes with an integrated SPARQL processor. To be able to access the data from outside, the data store also provides an HTTP server, which handles requests containing SPARQL queries to read the database, and requests containing N3 data to update the database. The SPARQL queries which are used to access the data are described in Section 6.3.4. These queries make use of a special OLAP ontology, to which an imported database has to be mapped, to access all relevant data for the NL processor and OLAP engine without the need to know the specific data fields. Thus, the same queries can be used for any database. The OLAP ontology and an example mapping is described in Section 6.3. The data store also provides a method to export the data in an XML format that can be read by the OLAP engine.

The *NL processor* is responsible for term extraction and taxonomy extraction. Two libraries are used for the basic operations to perform these tasks, namely GATE and NLTK. To access the data, the mentioned HTTP server, provided by the data store, and the predefined SPARQL queries are used. Thus, all text data in the database can be queried and subsequently processed. The result is then passed to the repository in N3 format, again via the HTTP server.

The *OLAP Engine* can import all the mapped and extracted information that is contained in the database. This information is stored in an OLAP cube on which the OLAP operations, including the Text Cube operations, can be performed.

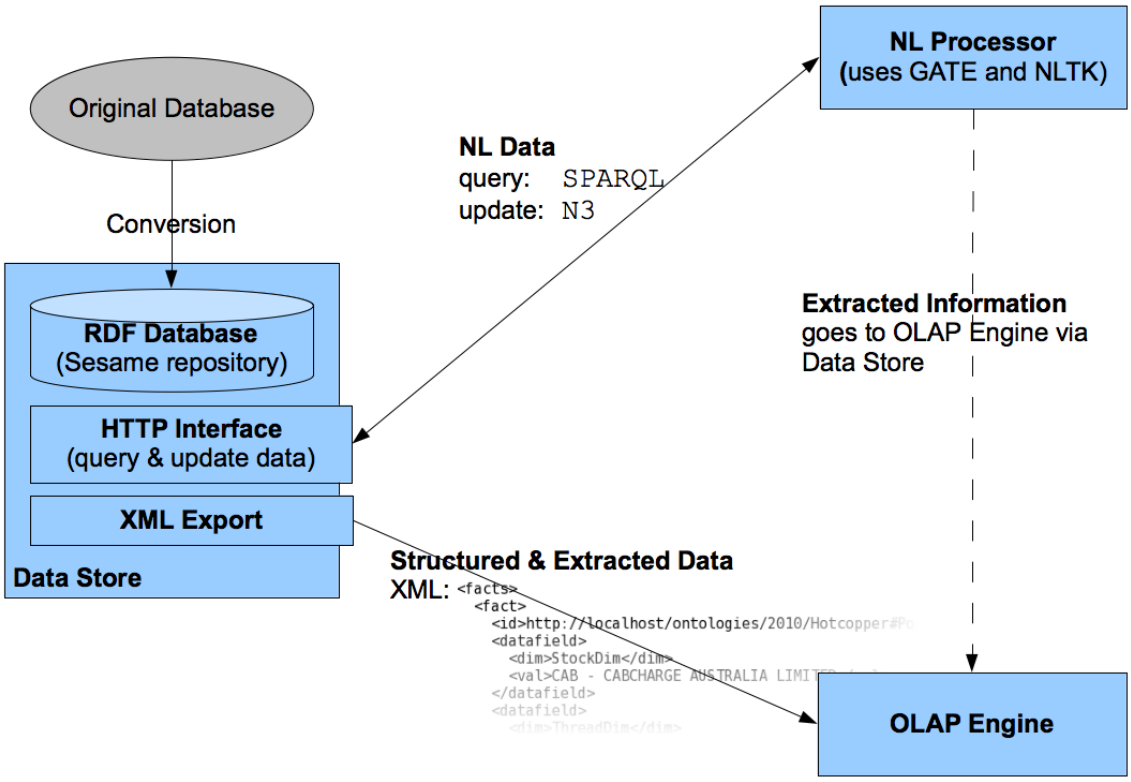

Figure 6.1.: Here the three main modules of the system and their interplay can be seen. The *Data Store* contains the Sesame RDF repository and the interface to access it via HTTP requests. Furthermore it provides a mechanism to export the data as XML so that it can be imported by the *OLAP Engine*. The *NL Processor* communicates with the data store via the HTTP interface by sending SPARQL queries to read the repository and by sending N3 data, to update the repository.

6.2. Natural Language Processing

The NLP tasks that have been performed were firstly term extraction, where in the following, term is used synonymously for *simple term*, which means terms consisting of only one token. Secondly, noun phrase extraction was performed and consequently relation finding between the noun phrases to create taxonomies.

The subtasks are performed by a number of different modules. The module that performs NP-chunking is written in Java and makes use of GATE Embedded as a programming library. As it is written in Java, it can directly access the datastore where it makes use of the predefined queries described in 6.3.4. The other tasks are performed by a number of Python modules, which partly make use of NLTK and access the datastore, to query and update the data, using the HTTP interface and also the predefined queries. In the following subsections, the single steps to perform the NLP tasks are described in detail.

6.2.1. Term Extraction

The first step is to fetch the data from the repository, which is done using the queries explained in chapter 6.3.4. Then for each text field, which will in the algorithm be referred as documents, the following algorithm is performed, using the according methods from NLTK as indicated:

```
1 for each document:
2     clean HTML tags          # nltk.clean_html(text)
3     tokenize the document    # nltk.word_tokenize(text)
4     remove non-alpha tokens
5     remove stopwords
6     stem the tokens          # porter.stem(token)
7     calculate the word frequency
8     for each token:
9         write according RDF data to repository
```

The cleaning of the HTML tags (line 2) is necessary because the text data can come from web sources, which is also the case for the example dataset. If there are no HTML tags in the text, this function doesn't change the document. In the following steps, the document is split into its tokens, and unwanted tokens like stopwords or numbers are filtered out. Before the term information is written to the RDF repository, the words get stemmed using the porter stemmer (van Rijsbergen et al., 1980).

With that process, the data for the information retrieval queries, as explained in Section 6.4.2.4, is extracted and can be used by the OLAP system.

6.2.2. Noun Phrases

In the prototype, GATE modules were used to extract the noun phrases. After the extraction process, the noun phrases were filtered, to keep only those which are potentially useful in the ontology building process. These steps are explained in more detail in the following subsections.

6.2.2.1. Chunking

Using GATE, the process of NP chunking is performed in three main steps (with the according GATE modules in brackets):

1. Sentence splitting (gate.creole.splitter.SentenceSplitter)
2. POS-tagging (gate.creole.POSTagger)
3. Chunking (mark.chunking.GATEWrapper)

The sentence splitter searches for typical sentence delimiters like "." or "?" and splits the text according to them. It also considers multiple delimiters like "?!" or recognizes if a "." is not used as a sentence delimiter but for instance in a floating point number.

The POS-tagger uses the resulting sentences and annotates each token in the sentence with part of speech tags like "CC" for *coordinating conjunction* or "NN" for *noun*. The tags are set according to rules which have been acquired by training on a labeled dataset.

For the chunking, the method described in Ramshaw and Marcus (1995) is used in the according GATE module. As a result, the noun phrases of each sentence are tagged, and can therefore be used for further processing. The exact documentation of these modules can be found on the GATE website[1].

6.2.2.2. Filtering

As the set of extracted noun phrases which is returned by the chunking process can be very large, it is necessary to filter the noun phrases, to reduce their number and only keep those, which are potentially useful for ontologies. Furthermore, if noun phrases come from an online database, like in the example dataset, where the content is generated in an informal way, as the input is not checked for correctness, the data can contain noise, like typing errors, which has to be filtered.

In the implementation the following filters have been applied, where the last two (*Commons* and *Dictionary*) filter noun phrases, that might be properly formed, but are not considered to be useful in the ontology building process:

- **Stopwords**: If a NP starts with one or more stopwords, these stopwords are removed from the NP. This is, to not get different versions of the same NP if it occurs with several determiners like "a" or "the".
- **Plurals**: If a NP is in the plural form, it is replaced by the singular form, to not get different versions of the same NP.
- **Numbers**: NPs that consist only of numbers are removed. Furthermore, NPs that start with a number and end with a plural "s" are reduced to their singular form.

[1] http://gate.ac.uk/

- **Commons**: NPs that appear frequently in several domains are considered as too general. For that purpose, the *Brown* corpus, which is available in NLTK and contains a set of documents where each document is assigned to one category, has been used. Thus, if a NP occurs in too many categories (according to a threshold), it is discarded.
- **Dictionary**: As the relation finding process that is later performed using the extracted NPs, is done by using a dictionary as background knowledge, NPs are only kept if they are in that dictionary. In the implementation Wikipedia was used as dictionary, thus only NPs which have an article in Wikipedia are kept.

The extracted noun phrases that pass these filter rules are then saved to the repository, including the information to which dimensions of which facts they are related. Therefore this information can later be used for the ontology building process, which is described in the next subsection.

6.2.3. Ontology Building

The noun phrases, that have been extracted, now serve as the basic concepts of the ontologies that are created. To build an ontology, it is essential, to relate the concepts in some way. The focus in this approach lies on finding paths between concepts via their categories, which can again be in higher categories. Therefore the resulting ontology is a tree where the nodes are associated with a child-parent relationship. Thus the resulting ontologies are so called lightweight ontologies or taxonomies, rather than general ontologies, which can have any type of relation between their nodes.

Therefore the steps of the ontology finding process are:

- **Selecting NPs**: First, a subset of all available NPs has to be selected on which the relation finding shall be performed. This is necessary, because bluntly relating all NPs, would consume too much time, as the number of pairs for n NPs is in $\mathcal{O}(n^2)$, and for each relation several online queries to the Wikipedia API have to be performed.
- **Relating NPs**: To relate the NPs, the categories for each one are extracted from the Wikipedia categorization system repeatedly. If the sets of categories of two NPs overlap, a relation between two NPs is found.
- **Extracting Taxonomies**: If pairs of NPs are related, these relations and their nodes are used to create a tree, by relating all possible nodes.
- **Pruning**: As the resulting trees can get too deep or too broad and can have only a small number of original NPs as their nodes, which can also be ambiguous, some pruning is performed to increase the quality of the resulting taxonomies.

These steps are explained in detail in the following subsections.

6.2.3.1. Noun Phrase Selection

The goal of the noun phrase selection is to get small subsets of the set of all noun phrases, where the items of a set are assumed to be related.

The first step is, to associate a *score* to each NP, according to its number of occurrences and the number of other NPs that co-occur with that NP. Thus, the score of a noun phrase p is $score(p) = \frac{\text{number of NPs that co-occur with } p}{\text{number of occurenes of } p}$. Therefore, a NP gets a higher score if it has many co-occurrences, and a lower score if it occurs more often. The motivation for this is, that an NP which occurs frequently, is considered to be more general than those, that have less occurrences and is therefore considered to be less "interesting" as a concept in an ontology. On the other hand, if an NP has many co-occurrences, it is considered to be more likely that a relation between the NP and one or more of its co-occurrences can be found, and therefore gets a higher score.

If the score for each NP is calculated, the list of NPs gets sorted descendingly according to the score. The top elements of the resulting list now serve as seed terms to find the previously mentioned subsets.

Here again the relation between NPs is considered. The relation between two NPs depends on the number of occurrences of each NP and the number of how often they co-occur. Thus, the "relatedness" of two NPs p_1 and p_2 is given by $rel(p_1, p_2) = \frac{num_{co}(p_1,p_2)}{num(p_1)} * \frac{num_{co}(p_1,p_2)}{num(p_2)}$, where $num(p)$ is the number of occurrences of p and $num_{co}(p_1, p_2)$ is the number of co-occurrences of p_1 and p_2. Thus the maximum value of "relatedness" is 1 if the two NPs always occur together, and 0 if they never co-occur.

The sets of NPs are then acquired with the following algorithm where n_max is the number of elements, the final set should have:

```
1 SET = {NP with the highest score which is not yet in any subset}
2 while size of SET < n_max:
3     rel_max = 0
4     for each p in NP that is not yet in any set:
5         rel = sum of relatedness values between p and elements of SET
6         if rel > rel_max:
7             rel = rel_max
8             candidate = p
9     add candidate to the SET
```

To further increase the relatedness between the elements of the set, the least related element can be removed and replaced by a new element, which has a higher sum of relatedness than the old value. Such elements can exist, because the elements are added one by one and therefore it cannot be guaranteed, that an added element is still among the highest related elements in the final set.

In the implementation, this algorithm was applied several times with $n\_max = 25$ to get a number of sets for the relation acquisition process, which is described in the following subsection.

6.2.3.2. Relation Finding

The relation acquisition process utilizes the categorization system of Wikipedia[2] to find relations between extracted noun phrases. As mentioned before, this is the reason why noun phrases are only kept in the filtering process if they have an associated article in Wikipedia.

In the categorization system, each article is associated with one or more categories, which in turn can be subcategories of other more general ones. For instance, the article for RAM is in category *Computer Memory* which is in turn in the category *Computer Hardware* and the article for CPU is in the category *Central processing unit* which also is in category *Computer Hardware* (among others). Therefore, the concepts RAM and CPU can be related via the path `RAM -> Computer Memory -> Computer Hardware <- Central processing unit <- CPU`, where *Computer Hardware* is the common parent.

Using the Wikipedia API[3] the categories of an article can be queried automatically. However, there are a few technical difficulties that have to be handled:

- **Number of queries restriction**: It is only allowed to send one query per second to the API, else the client IP address can be blocked. As the number of necessary queries can become quite big, especially for ambiguous items, redundant queries should be avoided.
- **Non-content categories**: The categorization system also contains categories which are used for maintenance, like *Category:Articles needing cleanup*, and are not related to the content of an article. Therefore, categories of that kind must be ignored in the relation finding process.
- **Ambiguous names**: Many names or concepts can have several meanings depending on the context. Thus, if a page title is ambiguous, a disambiguation page is provided, that links to all possible meanings. However, the API returns all links that are on the page which is displayed if it is opened in the browser. Therefore, it is not possible to bluntly follow any link that is returned, but only those where the link title is similar to the original title.
- **Redirections**: Some articles can be reached via several (similar) titles or capitalizations, where the content is associated with only one title, and the other titles just get redirected to that one. Therefore, the API can return a redirection page with the link to the title which is associated to the content. In that case this redirection has to be followed to obtain the categories.
- **Capitalization**: If a page request is sent, the title is transformed into a normalized form. However, different capitalizations can lead to different results in that process. This can be a problem, especially for user generated data like in the sample dataset where the input is not checked for correctness and the writing style is informal. Therefore, if a title can not be found with a certain capitalization, it can be useful to try several possible versions of it.

[2] http://en.wikipedia.org/wiki/Special:Categories
[3] http://en.wikipedia.org/w/api.php

In the implementation, the function that performs the API calls for a certain item, returns a set of possible article titles including the associated categories for each.

So the first step to relate pairs of noun phrases is to query their categories to a certain level, depending on how the maximum length of the resulting path between two noun phrases is set while also the relation between categories and their parents has to be kept. Thus for two noun phrases p_1 and p_2 a relation exists, if their sets of parents (of all levels) $par_{all}(p_1)$ and $par_{all}(p_2)$ are not distinct. To find the relations, it is only necessary to follow the paths from the common parents $par_{all}(p_1) \cap par_{all}(p_2)$ to the nodes p_1 and p_2 which represent the noun phrases.

This process is repeated for all pairs of a given set of noun phrases. The resulting relations are then used to create taxonomies, which is described in the next subsection.

6.2.3.3. Taxonomy Extraction

If for a set of noun phrases the relations between the pairs of its elements are found, these relations can be used to create taxonomies, which in turn contain the related noun phrases as leaf nodes. In fact a single relation can already be seen as a tree with two branches, ending in two leafs which represent the original noun phrases, where the root is the common parent.

Two such trees can be merged if they have one or more common nodes. In that case, the resulting graph simply inherits all nodes and edges of the two trees. The new graph is not necessarily a tree, as can be seen in the example in Figure 6.2.

After repeatedly applying this merge process on all relations and resulting graphs, until no more graphs can be merged, an initial ontology is achieved. If all nodes of the original relations are now considered as part of the ontology, the ontology is in general neither connected nor a tree. Therefore, it can be tried to connect nodes, which have no parents with the same process by which the noun phrases have been related earlier.

From the resulting graph, it is now possible to extract trees by declaring the nodes, which have no parents, as roots. For each of these roots, a tree can be extracted by taking only the nodes where a path from the root to the node exists in the directed graph. Thus the resulting trees can be seen as taxonomies and can therefore be integrated in the OLAP system.

As some, or even many, of these taxonomies might not contain a significant amount of nodes which represent original noun phrases, only those are kept, where this number is greater than a threshold. Furthermore, the resulting taxonomies can be too deep or too broad or have outlying nodes. For this reason pruning is performed on the trees, to increase the quality of the resulting taxonomies. The pruning process is described in the next subsection.

6.2.3.4. Pruning

The pruning steps that are performed on the extracted taxonomies deal with outliers, too long branches or branches that do not end in original noun phrases. In the following, nodes which have been derived from original noun phrases are denoted as *NP-node*.

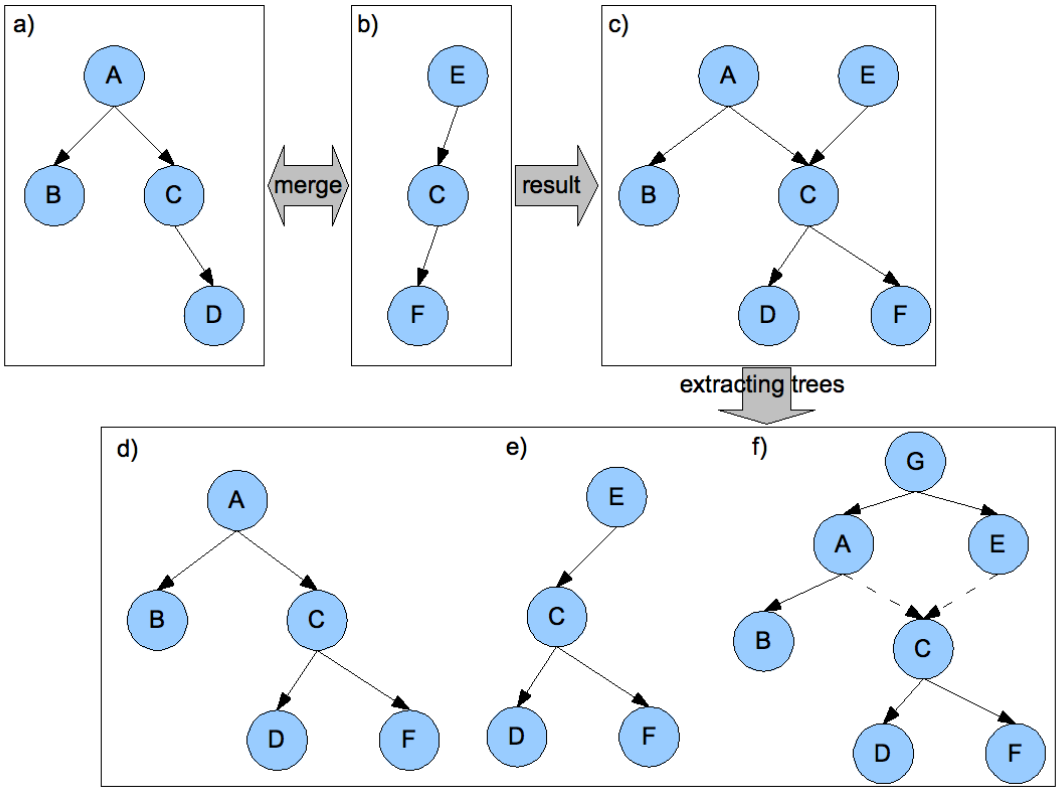

Figure 6.2.: If two trees are merged (a and b), the result (c) is not necessarily a tree. For this reason, either a relation between the parent nodes (A and E) has to be found, or a part of the graph must be omitted when a taxonomy is extracted. Thus the resulting trees can be either d), e) or f) if a relation between A and E can be found. In that case one of the edges between A/E and C has to be omitted.

Outliers and outlying subtrees: If the distance from an NP-node to the next nearest NP-node is greater than a threshold, the node is removed, unless another leaf with the same meaning of the same noun phrase, where this constraint does not apply, exists somewhere else in the tree.

A similar case is if the NP-nodes of a whole subtree are too far away from NP-nodes of outside the subtree. In that case the whole subtree is removed.

Branches not ending in a noun phrase: Because in practice it can not be guaranteed that the initial ontologies do not contain loops, the derived taxonomies may contain branches that do not end in an NP-node. Such branches are removed, as they are useless in the OLAP system.

Ambiguous meanings for the same noun phrase: Because noun phrases can have multiple meanings, as described earlier, they can also occur in one tree with several meanings. In this case, only the one meaning is kept, which is closest to other NP-nodes. If the distance for different meanings id equal, it is up to the user, which one should be kept.

Clipping nodes: As a final pruning step, nodes that have only one child can be removed. Thus the according children get connected to their parent's parent. The reason for this step is to make the taxonomy more convenient to process with pull up/push down methods in the OLAP process, because these operations on such nodes wouldn't change the OLAP cube.

The first three of these tasks are performed repeatedly until no more changes occur. After that the fourth task is performed. The resulting taxonomy can then be exported in a format, such that it can be integrated into the OLAP system, as described in chapter 6.3.2, to serve as an additional dimension.

6.3. Data Storage

In this Section the general OLAP ontology, including the queries which are used to obtain the data of a mapped database, is described. For the realization of the datastore a Java program was implemented that uses the Sesame library to create and access RDF triple stores. The datastore also provides an HTTP interface which handles requests to query and update the data. Furthermore, it provides methods to export the data as XML files which can in turn be imported by the OLAP engine.

6.3.1. The Dataset

For the experiments, data from the *Australian Stock Forum* (HotCopper[4]) was used. The database came in XML format and was imported into an RDF repository. An example of the original XML format is given in the following listing:

```
<hotcopper date="2008/11/13 19:12:404">
[...]
<post>
   <url>post_threadview.asp?fid=1&amp;tid=50000#124404</url>
   <thread>50000</thread>
   <id>124404</id>
   <suspended>None</suspended>
   <timestamp>15/08/03 13:00</timestamp>
   <sentiment>None</sentiment>
   <disclosure>No Stock Held</disclosure>
   <views>201</views>
   <stock>QAN - QANTAS AIRWAYS LIMITED</stock>
   <reply>Reply to:#124393 from clacoste </reply>
   <user>frankiedee</user>
   <title>re: qan (frankiedee)</title>
   <message>
      <div id="intelliTXT">clacoste,<br/>
         <br/>if it does take off (pardon the pun), I've got a very good idea where
              its going.<br/>
         <br/>I've already let the guys know that are part of AMTRADE.<br/>
         <br/>http://datafeeds.com.au/whocares.html</div>
   </message>
</post>
[...]
```

[4] http://www.hotcopper.com.au/

First the datafields have been identified and an ontology that describes the structure has been created. The data is organized as postings, where each posting belongs to a thread and has the following datafields:

- **url**: the URL of the posting
- **thread**: the thread to which the posting belongs to
- **id**: a unique number for each posing
- **suspended**: indicates the status of the author
- **timestamp**: the time when the posting was created
- **stock**: the stock of which the posting is about
- **sentiment**: indicates the sentiment of the author (e.g. buy, sell, etc.)
- **disclosure**: indicates if the author holds the according stock (optional)
- **views**: number of how often the posting has been viewed
- **reply**: the id of the posting to which the posting repies to
- **user**: the author of the posting
- **title**: the title of the posting
- **message**: the content of the posting

Thus the according ontology contains the following classes: Post, Thread, Author, SentimentValue, Stock, TimeYear, TimeMonth, TimeDay. Actually, the timestamp could have been mapped to the $xsd : dateTime$ data type but has been transformed to this representation to have an example for a hierarchically organized dimension which can be used in the OLAP system later.

The classes belong to the according fields and are related using object properties like for instance an author is related to a posting via the *hasAuthor* property. This is in RDF/XML representation:

```
<owl:Class rdf:about="#Post">
    <rdfs:label>Post</rdfs:label>
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasAuthor"/>
                    <owl:onClass rdf:resource="#Author"/>
                    <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger
                        ">
                        1
                    </owl:qualifiedCardinality>
                </owl:Restriction>
                ...
            </owl:intersectionOf>
        </owl:Class>
```

```
16      </owl:equivalentClass>
17  </owl:Class>
```

which means, that an instance of the class *Post* has exactly one relation to an instance of class *Author*.

The values for which no class exists are related to the according classes with data properties. For instance, the number of views of a posting is associated to the posting via the *viewCount* property, which contains an integer value. Again in RDF/XML representation this is:

```
1  <owl:Restriction>
2      <owl:onProperty rdf:resource="#viewCount"/>
3      <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">
4          1
5      </owl:qualifiedCardinality>
6      <owl:onDataRange rdf:resource="&xsd;integer"/>
7  </owl:Restriction>
```

A visualization of the resulting ontology is shown in Figure 6.3. In 6.3.3 and 6.3.2 will be described how this ontology is mapped to the general OLAP ontology.

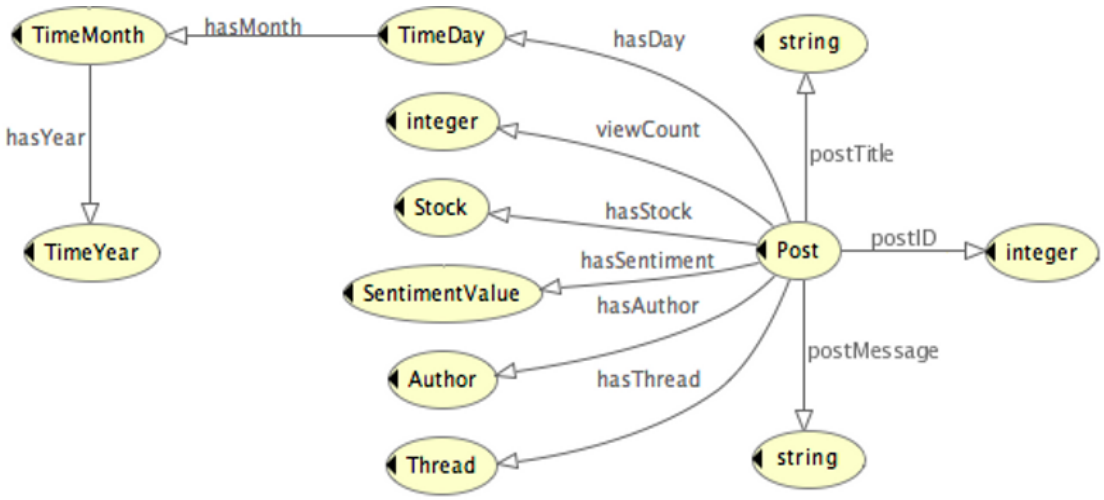

Figure 6.3.: The ontology of the example dataset. The *Post* class will later become the *Fact* in the OLAP ontology, and the connected classes that contain the related data will become the dimensions.

6.3.2. OLAP Ontology

The OLAP ontology describes a general representation of OLAP facts which are related to several dimensions. Furthermore, the developed ontology also contains classes to represent terms and taxonomies which consist of noun phrases that are extracted from the natural language data in the database and related using the methods described in chapter 6.2. A visualization of the ontology is shown in Figure 6.4. The classes and relations are described in the next subsections.

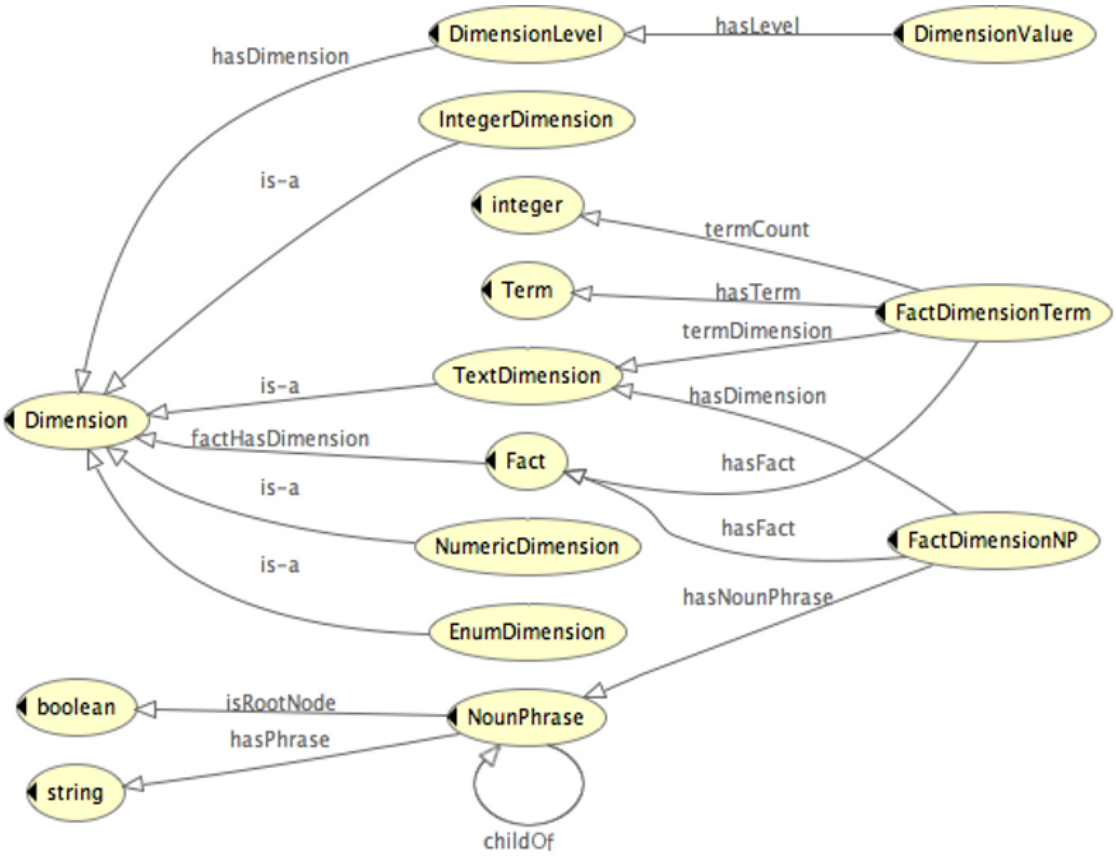

Figure 6.4.: The general OLAP ontology. The *Fact* class is related to the dimensions via the *factHasDimension* object property. The different types of dimensions are derived from the *Dimension* class. The classes *FactDimensionTerm* and *FactDimensionNP* connect the facts and the dimensions with the according terms and noun phrases, respectively.

6.3.2.1. Facts and Dimensions

In the ontology, facts are represented by the class $Fact$ and are related to the dimensions with the object property $factHasDimension$. Thus a fact can be related to any number of dimensions depending on the actual dataset.

The dimensions can be of the following types:

- **IntegerDimension**: A dimension containing an integer value, like for instance the number of inhabitants of a country or the quantity of a product in a sales record.
- **NumericDimension**: A dimension containing a floating point number, like for instance a temperature.
- **EnumDimension**: A dimension containing values that can be enumerated but probably not ordered. These dimensions can be structured hierarchically like for instance a geographic dimension with the hierarchy: *Countries* contain *States* contain *Cities*. Another case would be a time dimension where *Years* contain *Months* contain *Days*. The difference here is, that

the dates can be ordered in contrast to the values in the geographic hierarchy. Thus the time dimension could also be mapped to an *IntegerDimension*.

- **TextDimension**: Dimensions of this type contain natural language text. Therefore, they can not be used for OLAP directly, but the according information like terms or noun phrases contained in a certain value has to be extracted first. How extracted terms and noun phrases are related to the according datafields is explained in the following subsection.

Furthermore, each value is also associated to a certain level of a dimension. In case of non-hierarchical dimensions, all values have the same level. For other dimensions, the values belong to exactly one level of the according dimension. In the example of the geographic dimension, there would be three levels, namely for *Countries*, *States* and *Cities*.

6.3.2.2. Terms and Noun Phrases

Terms and noun phrases are represented in the ontology by the classes *Term* and *NounPhrase*. A term can occur in any number of facts while each occurrence is related to a certain dimension of type *TextDimension*. Furthermore, the number of occurrences of a term in a value of *TextDimension* is defined with the *termCount* data property. This enables the possibility to run IR queries on the database like searching for keywords or calculating the most frequent terms of a certain set of facts and dimensions. To gather all the necessary information, the class *FactDimensionTerm* links the fact, the term, the dimension and the term count for each occurrence of a term in a dimension of a fact.

Noun phrases are associated with facts in a similar way to the terms but without the number of occurrences in a certain text value. Furthermore, noun phrases can be structured hierarchically, similar to values of an *EnumDimension*. Thus a noun phrase can have the special property *isRootNode*, which means, that the according noun phrase is the root of an extracted taxonomy if this property value is set to *true*. If a noun phrase is a child of another noun phrase, this is indicated by the *childOfNP* object property. With this information, the whole taxonomy can be extracted, and used as an additional hierarchical structured dimension in the OLAP system. The connection between a noun phrase, a fact and a dimension is represented by the class *FactDimensionNP*, in an analogical way as it is done for terms.

These values can be queried and thus made available for OLAP. How this is done is described in Section 6.3.4.

6.3.3. Mapping to the OLAP Ontology

As mentioned before, to make a specific dataset available for the OLAP system, it has to be mapped to the OLAP ontology. Therefore, the first step is to identify the class which represents the OLAP facts and the associated datafileds which should become OLAP dimensions and furthermore to specify their type.

The according classes have then to be mapped to the OLAP classes, which will now be illustrated using the previously explained Stock-Forum dataset, using the N3 notation. In the example, the namespaces for the OLAP ontology and the forum data are *olapdim* and *hot* respectively.

First the postings become OLAP facts by declaring the *Post* class as a *Fact* with the statement: `hot:Post a olapdim:Fact .`

Now the following datafields shall become OLAP dimensions:

- **thread**: *IntegerDimension*
- **views**: *IntegerDimension*
- **timestamp**: *EnumDimension*
- **stock**: *EnumDimension*
- **sentiment**: *EnumDimension*
- **disclosure**: *EnumDimension*
- **user**: *EnumDimension*
- **title**: *TextDimension*
- **message**: *TextDimension*

Thus for each of these data fields a dimension has to be created. This is illustrated with the following examples.

The statement `:ViewsDim a olapdim:IntegerDimension.` creates a class for dimension *views* with name *ViewsDim*. The level *ViewsLevel* for this dimension is created with

```
1 :ViewsLevel a olapdim:DimensionLevel ;
2     olapdim:hasDimension :ViewsDim ; olapdim:subLevelOf :ViewsLevel .
```

Here the level is declared as a sublevel of itself, which means it is the highest level of a hierarchy.

The next step is to declare the value for the dimension. If the dimension comes from a data property, like in this case, a new class has to be created. Else the class to which the fact is associated to becomes the *DimensionValue*. In this case, an instance *ViewCount* of *DimensionValue* is created and connected to the actual value property *viewCount*. This is done with the following statements:

```
1 :ViewCount a olapdim:DimensionValue ; olapdim:hasLevel :ViewsLevel .
2 hot:viewCount a olapdim:dimensionValue ; rdfs:Domain :ViewCount .
```

Because *viewCount* is a data property in the original database which is connected to *Post*, the following statement is necessary to connect the fact with the dimension:

```
1 hot:Post a :ViewCount .
```

For dimensions, that are derived from object properties, rather than data properties, this last step is not necessary.

To also give an example of an *EnumDimension* which comes from a object property, the declaration for the stock dimension consists of the following statements:

```
:StockDim a olapdim:EnumDimension .
:StockLevel a olapdim:DimensionLevel ;
    olapdim:hasDimension :StockDim ; olapdim:subLevelOf :StockLevel .
hot:Stock a olapdim:DimensionValue ; olapdim:hasLevel :StockLevel .
hot:stockName a olapdim:dimensionValue ; rdfs:Domain hot:Stock .
```

Despite the time dimension, all other dimensions are created analogical to these examples. The difference to the time dimension is, that it has three levels and is therefore hierarchically structured. The basic declaration of the dimension however looks the same:

```
:TimeDim a olapdim:EnumDimension .
```

But as it has three levels, these are declared with the following statements, for year, month and day respectively:

```
:TimeYearLevel a olapdim:DimensionLevel ;
    olapdim:hasDimension :TimeDim ;
    olapdim:subLevelOf :TimeYearLevel .
:TimeMonthLevel a olapdim:DimensionLevel ;
    olapdim:hasDimension :TimeDim ;
    olapdim:subLevelOf :TimeYearLevel .
:TimeDayLevel a olapdim:DimensionLevel ;
    olapdim:hasDimension :TimeDim ;
    olapdim:subLevelOf :TimeMonthLevel .
```

Furthermore, in the database the values are structured in a way, such that a month is assigned to a year and a day is assigned to a month. Therefore, values can be children of other values. This information is mapped to the OLAP ontology by declaring the *childOf* property for the according *DimensionValue* instances:

```
hot:TimeYear a olapdim:DimensionValue ; olapdim:hasLevel :TimeYearLevel .
hot:TimeMonth a olapdim:DimensionValue ;
    olapdim:hasLevel :TimeMonthLevel ; olapdim:childOf hot:TimeYear .
hot:TimeDay a olapdim:DimensionValue ;
    olapdim:hasLevel :TimeDayLevel ; olapdim:childOf hot:TimeMonth .
```

Furthermore, the property which relates two values in the database hierarchically has to be mapped to the *childOf* property:

```
hot:hasYear a olapdim:childOf .
hot:hasMonth a olapdim:childOf .
```

And finally, the data properties which hold the actual values have to be declared:

```
hot:yearValue a olapdim:dimensionValue .
hot:monthValue a olapdim:dimensionValue .
hot:dayValue a olapdim:dimensionValue .
```

Again, this works analogic for any kind of hierarchically structured dimension.

With such a mapping, all desired values of a database and especially the hierarchies can be queried using general queries, that only know the OLAP dimension, but don't have to be modified for different databases. These queries are shown in the next subsection.

6.3.4. Queries

If a database is mapped properly to the OLAP ontology and the terms and noun phrases are extracted from the natural language data, the data can be queried with predefined SPARQL queries and therefore be brought into the OLAP system. The specific queries are explained in the following subsections.

6.3.4.1. Querying Dimensions

Now the goal is, to query each dataset which is a fact with the according datafileds and their dimensions and furthermore the information of which type a dimension is. So the first step is to get all available dimensions or rather their associated *dimensionValue* including the child-parent information. This is done with the following SPARQL query:

```
SELECT ?dimVal ?child ?dim ?label ?type
WHERE {
  ?dimVal a olapdim:dimensionValue .
  ?dimVal rdfs:domain ?DV .
  OPTIONAL {
    ?child olapdim:childOf ?DV .
  }
  ?DV olapdim:hasLevel ?level .
  ?level olapdim:hasDimension ?dim .
  ?dim a ?dimtype .
  ?dimtype rdfs:label ?type .
  OPTIONAL {
    ?dim rdfs:label ?label .
  }
}
```

This returns all *dimensionValue* instances in the variable *?dimVal*. The *?child* variable is optional and, if set, holds the instance of *DimensionValue*, which is a child of the *DimensionValue* which is associated with *?dimVal*. Furthermore, *?type* holds the type of the dimension of the value, which can therefore be one of *IntegerDimension*, *NumericDimension*, *EnumDimension* or *TextDimension* (or rather the label of these dimensions) and *?dim* is the dimension defined in the mapping. If

this query is executed on the database with the mapping, which was introduced in the previous Section, one row for the year value of the time dimension has the following values:

- **?dim**: *TimeDim* (the dimension for the time, which was defined in the mapping)
- **?dimVal**: *yearValue* (which is the data property for the year in the original database)
- **?child**: *TimeMonth* (which is a *DimensionValue* and is the child of *TimeYear*)
- **?label**: *None* (as the label is not set in the example)
- **?type**: *EnumDimension* (which is set as the label of the enum dimension in the OLAP ontology)

In the next subsection it is explained how this information is used to query the facts and the values that are related to them.

6.3.4.2. Querying Facts

To simply get the values which are facts can be done with the following query:

```
1  SELECT ?fact
2  WHERE {
3    ?factC a olapdim:Fact .
4    ?fact a ?factC .
5  }
```

This returns a list of all instances of *?factC* which is in turn an instance of the *Fact* class in the OLAP ontology. Thus, in the example database, where the postings have been defined to be facts in the mapping by the statement `hot:Post a olapdim:Fact .`, this returns a list of all instances of the *Post* class in the database.

These instances are now related to the according values by the properties that have been mapped to the *dimensionValue* properties in the OLAP ontology. Thus, with the result of the query of the previous Section, the actual value for each dimension for a certain fact can be queried. For this purpose, another query, where the fact and the dimension is defined, is executed to get the according value. This is done with the following query, where *fact* is the URI for the fact, which is obtained from the previous query, and *dim* is the URI for the *dimensionValue* instance, which was obtained by the query in the previous Section:

```
1  SELECT ?dim
2  WHERE {
3     {<fact> <dim> ?dim . }
4     UNION
5     {
6     <fact> ?anypredicate ?dimVal .
7     ?dimVal a ?DVC .
8     ?DVC a olapdim:DimensionValue .
9     ?dimVal <dim> ?dim .
10    }
11 }
```

Because the values can come from either a data property or an object property, there is always only one of the possibilities of the *UNION* satisfied and therefore the proper value is returned.

If such a query is executed for each of the available dimensions, all values for a fact can be obtained. Another way would be to dynamically construct a query for all dimensions at once, but in the implementation this turned out to be slower due to the higher complexity of the query.

What remains now is to query the terms and noun phrases which are associated to a certain fact which is explained in the next subsection.

6.3.4.3. Querying Terms and Noun Phrases

The terms and noun phrases for a fact can be obtained by a single query, as they are associated to facts via the *FactDimensionTerm* class or the *FactDimensioNP* class, respectively.

With the following query the terms for a certain fact can be obtained:

```
SELECT ?term ?dim ?freq
WHERE {
  ?fdt a olapdim:FactDimensionTerm .
  ?fdt olapdim:hasFact <fact> .
  ?fdt olapdim:termCount ?freq .
  ?fdt olapdim:termDimension ?dim .
  ?fdt olapdim:hasTerm ?term .
}
```

If this query is executed, the result set contains a row for each term occurrence of that fact, where the variables *?term*, *?dim* and *?freq* contain the term URI, the dimension URI and the number of occurrences (integer) respectively. The query for the noun phrases works similarly, only that it has no property for the number of occurrences but additional statements to make sure that a noun phrase is part of an taxonomy, as others can not be utilized in the OLAP system. These statements are:

```
  ?np olapdim:childOfNP ?parent .
  ?np olapdim:hasRoot ?root .
```

Thus, the noun phrase with the URI *?np* must be a child of another noun phrase.

To query all nodes of extracted taxonomies, first the root nodes are queried, which is simply done by selecting all instances of *NounPhrase* where the *isRootNode* property is set to *true*. Then, for each root node, all child nodes can be obtained with:

```
SELECT ?np ?label
WHERE {
  ?np a olapdim:NounPhrase .
  ?np rdfs:label ?label .
  ?np olapdim:childOfNP <nounphrase> .
}
```

where *nounphrase* is the parent of the returned noun phrases. If this query is applied recursively, until no more children are returned, a whole taxonomy can be obtained. Therefore, this process has to be done for each root node.

With the queries which have been described in this Section, all information can be extracted from the repository, and therefore be made available for the OLAP system. How the system uses that information for the typical OLAP function is described in chapter 6.4.

6.3.4.4. Querying Text Data for NLP

Terms and noun phrases fist have to be extracted from the according data in the dataset to make them available for the OLAP system. Therefore, all the datafields that contain natural language data have to be queried to pass the data on to the module of the system, which is responsible for the NLP tasks.

As datafields that contain NL-text are of type *TextDimension*, this can be done using similar queries to those described before. It only has to be ensured that the information, to which fact and dimension some text data belongs, is kept. Thus, for a fact with URI *facturi*, the text data with the according dimensions can be obtained with the query:

```
1  SELECT ?text ?dim
2  WHERE {
3    ?dim a olapdim:TextDimension .
4    <facturi> ?dimval ?text .
5    ?level a olapdim:DimensionLevel .
6    ?level olapdim:hasDimension ?dim .
7    ?dimVal a olapdim:DimensionValue .
8    ?dimVal olapdim:hasLevel ?level .
9    ?dimval a olapdim:dimensionValue .
10   ?dimval rdfs:Domain ?dimVal .
11 }
```

where *?dim* has to be a *TextDimension* as stated in the first line of the *WHERE*-block, and *?text* is a value that is associated with the specified fact. The remaining lines ensure, that the value *?text* is the associated *dimensionValue* for the dimension *?dim*.

6.4. OLAP Engine

The following subsections describe the implementation of the OLAP engine and the according methods and how they are performed in detail. This part of the system is implemented as a separate C++ program, that can import the XML files that are created by the datastore.

6.4.1. OLAP Cube

As the values of these dimensions define the cells in the cube, the dimensions of a cell can be seen as a set of constraints, which filter the facts which belong to the cell if applied to the set of all possible facts. Typically, the dimensions of the cube are hierarchically structured dimensions, such that the drill up/down methods, can be applied. But with the constraint interpretation it is also possible to add numeric dimensions to the cube and restrict the values of the according cells to certain values or intervals. The disadvantage of allowing such dimensions is, that the computation can become more complex and furthermore that the drill up/down methods can not be applied on these dimensions. However, to allow these dimensions can help to analyze the dataset because it offers additional possibilities to materialize the cube.

6.4.2. Implemented OLAP and Text Cube Methods

In this subsection, the OLAP methods and aggregation functions, which have been implemented in the prototype, are described.

6.4.2.1. Aggregation Functions

Aggregation functions are functions like *sum* or *average*, that are applied on cells of the cube on a dimension which holds the appropriate type of value.

The following aggregation functions for numeric dimensions have been implemented in the prototype:

- **sum**: the sum of the values of a certain dimension
- **avg**: the average of the values of a certain dimension
- **min**: the minimum value of a certain dimension
- **max**: the maximum value of a certain dimension

where all of these functions can be applied to dimensions of type *IntegerDimension* and *NumericDimension*, which have been defined in the OLAP ontology in chapter 6.3.2. Therefore, the according dimensions represent possible measures in the cube. Furthermore, two aggregation functions for text dimensions have been implemented:

- **most frequent terms**: This aggregation function calculates the most frequent terms of a textual dimension for each cell of a cube.
- **keyword search**: Given a set of keywords, this function returns all documents of a cell that satisfy the keywords.

6.4.2.2. Drill up/down

The drill up and drill down functions can be performed on hierarchically structured dimensions. Therefore, each value v_i of a dimension d is associated with exactly one level l_k of that dimension.

Drill up: Given a cube C with dimension d which is currently on level l_k means, that dimension d of C consists of values which belong to l_k. If l_k is not already the highest level in the hierarchy, each value has exactly one parent. If *drill up* is performed on d, all values of l_k are removed and replaced by their parents, which therefore are of level l_{k-1}.

Drill down: This is the reverse operation to drill up. Therefore, performing drill down on dimension d with level l_k removes all values of l_k and adds all their children instead. Thus, all values are now of level l_{k+1}.

The drill up and down functions therefore provide a method to view the dataset on different levels of detail with respect to a certain dimension.

6.4.2.3. Slice

With slice, the values of a certain dimension in a cube can be restricted such that the cube can be viewed in further ways.

Slice: Given the set V_d of current values of a dimension d of cube C. *Slice* can be performed on a $v \in V_d$ and reduces the according dimension d to only the value v. Thus, the resulting cube contains only facts that have value v of dimension d.

6.4.2.4. Term Queries

On a set of facts, two types of term queries can be performed:

- **Keyword search**: Given a set F of facts and a textual dimension d, where each fact $f_i \in F$ has a number of terms $T_d(i)$ associated with dimension d and a set Q of search terms. If the search is performed, the resulting set R consists of the facts, that have all terms $t \in Q$ associated with dimension d, thus $R = \{f_i \in F \mid \forall t \in Q : t \in T_d(i)\}$.
- **Most frequent terms**: Given a set F of facts, a textual dimension d and set $T = \{t \mid \exists f \in F : t \in T_d(i)\}$ (with $T_d(i)$ as before) which contains all terms that are associated with dimension d and facts in F. Let $n_t(f, d)$ be the number of occurrences of term t in dimension d of fact f and $N_t(F, d) = \sum_{f \in F} n_t(f, d)$ the sum of all occurrences of t in set F. Then the *most frequent terms* query returns the elements $t \in T$, sorted descendingly according to $N_t(F, d)$.

6.4.2.5. Pull-Up and Push-Down

The pull-up and push-down methods can be applied on dimensions, that are derived from extracted taxonomies, as follows:

- **pull-up**: Given a level L of nodes $n_1, \ldots, n_k$ and let $par(n)$ be the parent node of n and $des(n)$ the descent nodes of n. If pull up is performed on node $n_i \in L$, the new level L_{new} is achieved by removing all descent nodes of $par(n)$ from L and adding $par(n)$ to the set. Thus, $L_{new} = \{n \mid n \in L \wedge n \notin des(par(n_i))\} \cup par(n)$.
- **push-down**: Push-down is the reverse function of pull-up. Thus, given a level L with nodes $n_1, \ldots, n_k$, push-down on node n_i removes n_i and adds $chld(n_i)$, the set of children of n_i, to the new level $L_{new} = \{n \mid n \in L \setminus \{n_i\}\} \cup chld(n_i)$.

6.5. Results

In this Section the outcome of the taxonomy extraction using the previously described dataset is shown. Subsequently, the integration of the extracted information, namely taxonomies and term frequency, into the Text Cube is demonstrated. Furthermore, the operations that are made possible by integrating this additional information are shown.

6.5.1. Taxonomy Extraction

The first step of the taxonomy extraction is to extract the NPs from the dataset. Therefore, NP-chunking using GATE was performed on a set of about 50,000 postings. From the extracted NPs, those are considered as potential candidates for taxonomy concepts, which have a page in Wikipedia associated with them, and are not filtered by any other rules described in Section 6.2.2.2. After the according filtering process about 11,000 NPs remained as concept candidates.

After the filtering process, small subsets (clusters) of the NPs are calculated as described in Section 6.2.3.1. The hypothesis is, that a significant number of the NPs of a cluster, which are selected by this process, are of the same domain and can therefore be related to each other. As can be seen in the following examples, the resulting clusters do in fact tend to contain elements of a certain domain, although the number of elements that do not fit to the others is varying. The good clusters can then be used for the relation finding process, as described in Section 6.2.3.2.

```
This is an example for a good cluster, where most of the NPs are from the
medical domain:
['glaxosmithkline', 'neuraminidase_inhibitors', 'rsv', 'smallpox',
'infectious_diseases', 'common_cold', 'hepatitis_b', 'sankyo', 'immunotherapy',
'incorporation', 'eradication', 'coupling', 'superiority', 'lung', 'mechanism',
'protein', 'platform_technology', 'manufacturer', 'tissue', 'scotch',
'antigen', 'dendritic_cells', 'viral_protein', 'adjuvants', 'hbv']

This is another quite sensible example, as most of the terms are related to the
```

```
IT domain:
['cache', 'ku_band', 'narrowband', 'configurations', 'ka_band', 'effective',
'oceania', 'asiasat', 'microsoft_office_2003', 'verizon', 'microsoft_software',
'loral', 'information_technology', 'connectivity', 'analogue', 'sme', 'page_3',
'online_advertising', 'competitive_advantage', 'security_services', 'shanghai',
'sta', 'dividend_payout_ratio', 'negligence', 'wedo']

The following cluster is probably not well suited for relation finding.
Although it contains several elements that are of the domain economy, it also
contains several elements that don't seem to fit properly:
['dennis', 'nominee', 'arrogance', 'nyt', 'associated_press', 'fidelity',
'susan', 'coo', 'hordes', 'pension', 'high_time', 'rid', 'real_change',
'fraternity', 'economist', 'businessweek', 'national_association', 'buffett',
'case_study', 'adviser', 'embarrassment', 'advisory_board', 'proxy_fight',
'qwest', 'stanford']
```

The first of the example clusters is used for the relation finding, as most of the elements it contains are related to the medical domain and therefore they can be expected to yield a sensible taxonomy. In Figure 6.5 the initial taxonomy is shown. As can be seen, it still contains undesirable nodes and branches which are handled by pruning the taxonomy as described in Section 6.2.3.4. The resulting taxonomy can be seen in Figure 6.6.

The resulting taxonomy can now be exported in N3 format, such that it can be added to the RDF repository. This automatically adds the additional dimension which is represented by the according taxonomy. The operations that can be performed on this dimension are also shown in the following Section.

6.5.2. Text Cube

The Text Cube now offers both, the traditional OLAP functionality and the additional functionality defined by Text Cube. Thus, a possible application of the traditional OLAP functions on the example dataset could be for instance to instantiate the cube by using the time dimension for year *03* and the stock dimension by using some of its values. A possible aggregation function is now to count the number of datapoints of each cell. This can be seen in Figure 6.7.

If now slice is performed on value `VCR - VENTRACOR LIMITED` of the stock dimension followed and a drill down on the time dimension the values of the time dimension correspond to the months of the according year. An aggregation function can for instance also be applied on the views dimension to get the average number of views of the postings in each cell. The result of these operations can be seen in Figure 6.8.

As the Text Cube also defines information retrieval functionality, it is now possible for a list of keywords to firstly find those cells that contain documents that satisfy the list, and secondly to find the specific documents of the cube or a cell which satisfy the list. Thus, instead of performing a keyword search on the text only, the documents can be restricted, such that they satisfy certain requirements, like in the example being about stock `VCR - VENTRACOR LIMITED` and for dates as before. In Figure 6.9 the number of documents that satisfy the keywords *patient, heart* and *device*

can be seen. If now the documents of a specific cell are of interest, the cube can be sliced to that cell and a keyword search that returns the relevant documents can be performed like in Figure 6.10 for stock as before and date `08-03`. Therefore, not the whole set of documents has to be searched, and less non relevant documents are returned, due to the restrictions on the dimensions.

With the previously extracted *Health* taxonomy, the dataset additionally has the dimension defined by the taxonomy to those dimensions described in 6.3.3. Thus, the cube can now for instance be instantiated by considering the two dimensions *Health* and *time*. As aggregation function, the five most frequent terms shall be calculated for each resulting cell, which yields the result displayed in Figure 6.11.

If push down is now performed on *Health*, and then again on *Diseases and disorders*, the *Health* dimension contains the values *Category:Health effectors, Category:Health fields, Category:Infectious diseases* and *Category:Eradicated diseases* which yields the result displayed in Figure 6.12 (again with the most frequent terms as aggregation function).

Thus, a new dimension can easily be added, as it gets automatically queried when the dataset is imported into the OLAP engine. Furthermore, the most frequent terms as aggregation function provide a powerful extension for a document driven analysis with little structured information.

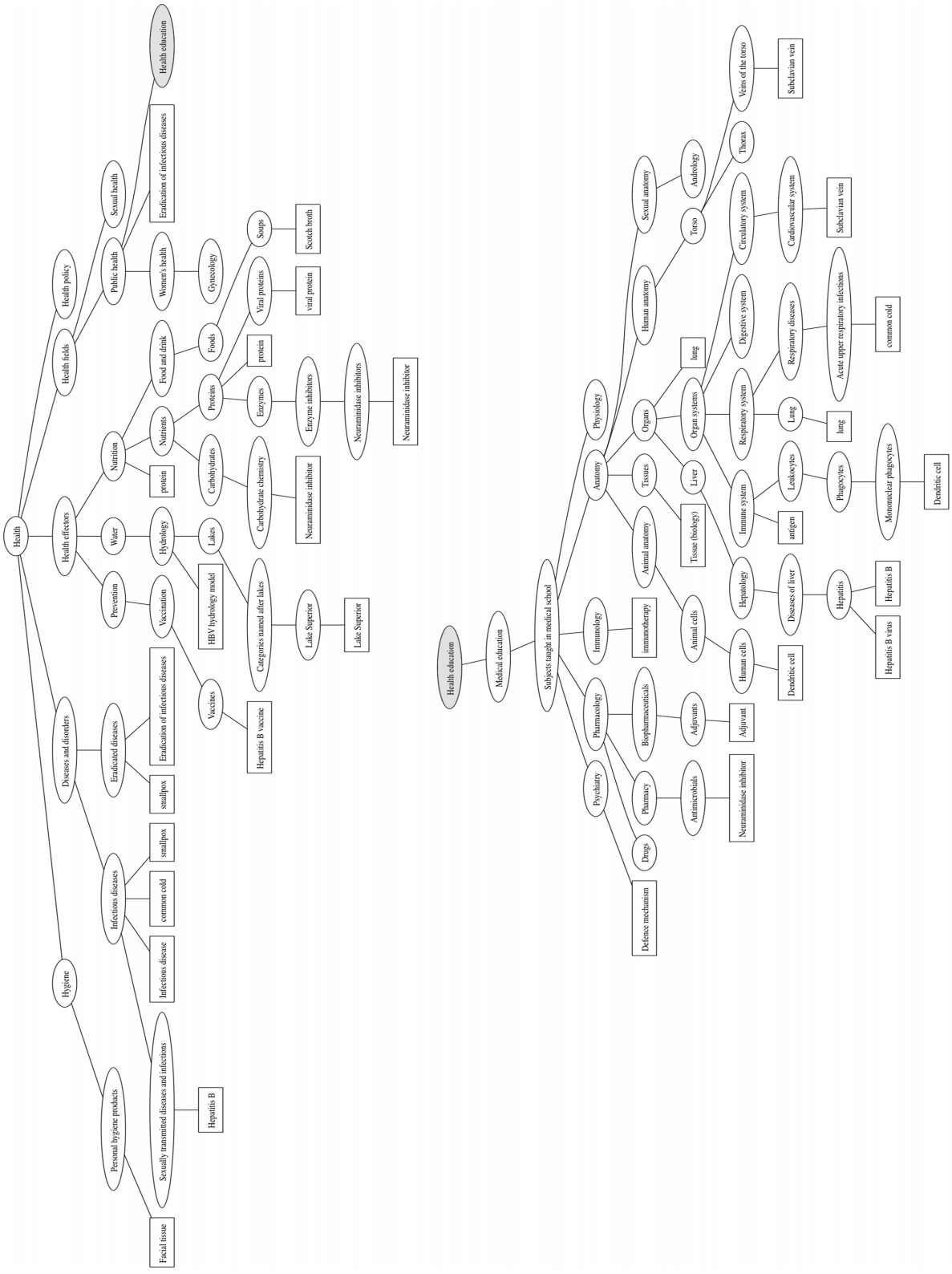

Figure 6.5.: Here an example of a not yet pruned taxonomy is shown. In the graphic it is split up into two trees at he colored node to fit on the page. The rectangular nodes represent original NPs, while the other nodes represent categories which were queried from Wikipedia. The labels of the NPs correspond to the titles of the associated Wikipedia articles. As can be seen, the taxonomy contains many undesirable elements like outliers or branches not ending in a original NP. In Figure 6.6 the same taxonomy after the pruning process is shown.

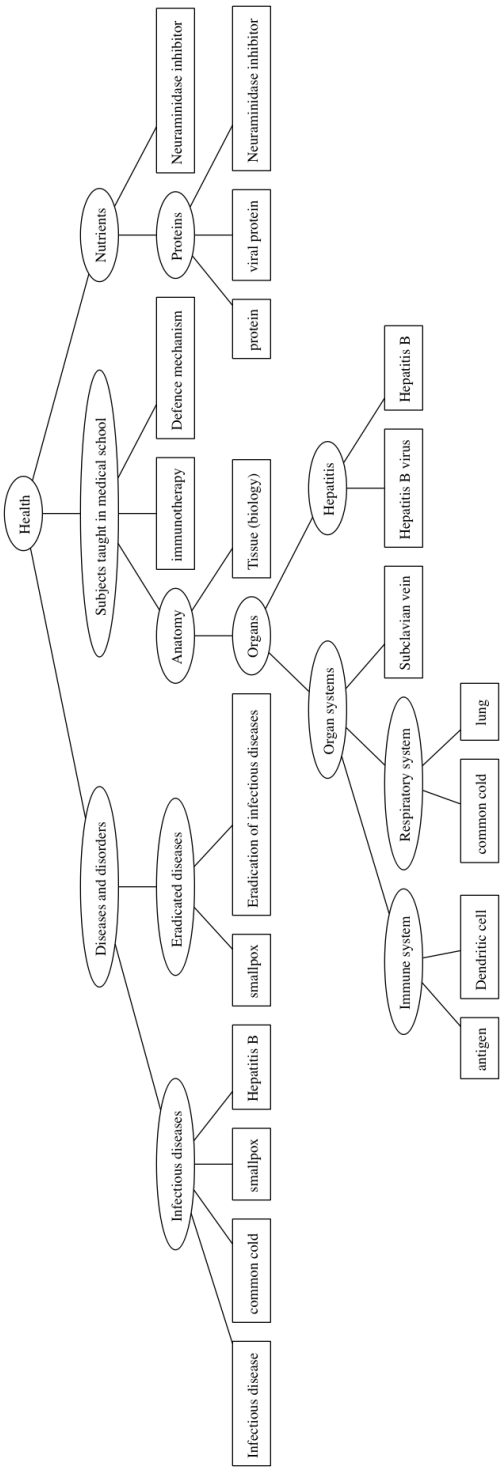

Figure 6.6.: Here the taxonomy of Figure 6.5 is shown after the pruning process. Again the rectangular nodes represent original NPs. As can be seen, in this taxonomy, the issues that are addressed in the pruning process are resolved.

	03
ADU - ADAMUS RESOURCES LIMITED	46
AFT - AFT CORPORATION LIMITED	127
ASX - ASX LIMITED	413
BDL - BRANDRILL LIMITED	436
COE - COOPER ENERGY LIMITED	200
GTG - GENETIC TECHNOLOGIES LIMITED	394
SLT - SELECT VACCINES LIMITED	920
VCR - VENTRACOR LIMITED	1314

Figure 6.7.: Here the cube is displayed, considering some values of the stock and time dimensions. The aggregation function is the number of the datapoints in an according cell.

	12-03	10-03	11-03	08-03	09-03
VCR - VENTRACOR LIMITED	0	101	45	815	353

(a) Cube with number of facts as aggregation function.

	12-03	10-03	11-03	08-03	09-03
VCR - VENTRACOR LIMITED	0	201.356	151.133	197.552	293.433

(b) Cube with average number of views as aggregation function.

Figure 6.8.: After a drill down on the time dimension and a slice on `VCR - VENTRACOR LIMITED`, the cube of Figure 6.7 has the cells shown in these examples. In subfigure 6.8a the aggregation function is the number of facts and in 6.8b it is the average number of views for each posting. The reason why not all months of the according year are shown is, that the used dataset doesn't contain datapoints with the according values.

	12-03	10-03	11-03	08-03	09-03
VCR - VENTRACOR LIMITED	0	0	0	20	9

Figure 6.9.: This example shows how many documents of each cell satisfy a given list of keywords.

	08-03
VCR - VENTRACOR LIMITED	http://localhost/ontologies/2010/Hotcopper#Post124849 http://localhost/ontologies/2010/Hotcopper#Post124853 http://localhost/ontologies/2010/Hotcopper#Post124901 http://localhost/ontologies/2010/Hotcopper#Post124909 http://localhost/ontologies/2010/Hotcopper#Post125614 http://localhost/ontologies/2010/Hotcopper#Post125616 http://localhost/ontologies/2010/Hotcopper#Post126498 http://localhost/ontologies/2010/Hotcopper#Post126533 http://localhost/ontologies/2010/Hotcopper#Post126692 http://localhost/ontologies/2010/Hotcopper#Post126768 http://localhost/ontologies/2010/Hotcopper#Post127309 http://localhost/ontologies/2010/Hotcopper#Post127921 http://localhost/ontologies/2010/Hotcopper#Post128513 http://localhost/ontologies/2010/Hotcopper#Post128711 http://localhost/ontologies/2010/Hotcopper#Post128718 http://localhost/ontologies/2010/Hotcopper#Post128752 http://localhost/ontologies/2010/Hotcopper#Post128801 http://localhost/ontologies/2010/Hotcopper#Post128889 http://localhost/ontologies/2010/Hotcopper#Post129738 http://localhost/ontologies/2010/Hotcopper#Post129813

Figure 6.10.: Here the documents of a cell, that satisfy a certain list of keywords are listed (here these are the ids of the according postings). The keywords for the example were *patient, heart* and *device*.

	Category:Health
12-03	
10-03	hiv (7) drug (6) said (5) treatment (5) effect (4)
11-03	trial (14) vaccin (13) australia (9) hepat (8) clinic (7)
08-03	dctag (36) vaccin (27) cell (19) technolog (17) result (16)
09-03	research (100) patent (84) test (69) jonathan (68) holm (68)

Figure 6.11.: This example makes use of the previously extracted taxonomy. The second dimension is the time dimension. The aggregation function is *most frequent terms*. The reason for the first cell being empty is, that it doesn't contain any facts. These are again the stemmed versions of the terms and the numbers denote the number of occurrences of the preceding term.

	Category:Nutrients	Category:Subjects taught in medical school	Category:Infectious diseases	Category:Eradicated diseases
12-03				
10-03		hiv (7) drug (6) said (5) treatment (5) effect (4)	hiv (7) drug (6) said (5) treatment (5) virax (4)	
11-03		vaccin (13) hepat (8) select (6) diagnost (6) test (5)	vaccin (13) hepat (8) select (6) diagnost (6) test (5)	trial (14) australia (6) solbec (6) clinic (6) lead (5)
08-03	dctag (36) vaccin (22) result (16) human (16) cell (14)	dctag (36) vaccin (27) cell (19) human (16) result (16)	dctag (36) vaccin (27) cell (19) technolog (17) result (16)	dctag (36) vaccin (22) technolog (17) human (16) result (16)
09-03	patent (80) jonathan (68) holm (68) dr (62) research (50)	research (58) develop (49) drug (45) compani (44) bioeffect (39)	biota (14) test (13) diagnost (11) compani (11) diseas (9)	biota (14) drug (8) test (7) agreement (7) sar (7)

Figure 6.12.: This example shows the cube of Figure 6.11 after performing two push down operations. These are again the stemmed versions of the terms and the numbers denote the number of occurrences of the preceding term.

7. Lessons Learned

The work on the thesis was a good opportunity to gain some insights into the fields that were treated in this work. In this Chapter, the most relevant things that came up in the phases of literature review, implementing and testing are discussed.

7.1. Literature Review and Academic Writing

As the implemented system consists of three main components - the datastore, the OLAP engine and the natural language processor - it was necessary to get into three fields of computer science, namely NLP, OLAP and semantic web technologies. Among those, the semantic web technologies are relatively easy to get into, as they are defined by a number of specifications of the W3C and therefore all necessary information is freely available and relatively easy to find. Furthermore, a broad spectrum of according software is available, such that it is quite straightforward, to set up an experimenting environment. However, as the semantic web technologies are relatively new, many available tools and libraries are not quite technically mature.

The fields of NLP and OLAP were actually not too easy to get into. Especially NLP is a field, that requires quite a lot of effort. As research in NLP has now been done for decades, the amount of literature is overwhelming. This makes it hard for a non-expert to get into it and find the literature, that is relevant for a specific task. Furthermore, even very basic problems of NLP, like segmentation or POS-tagging, are very hard to tackle, as these problems cannot be well defined formally. Thus, to really understand these problems in detail, it becomes necessary to understand the theories behind them, which would however consume too much time and thus, some solutions just have to be taken for granted.

The subfield of ontology extraction also bears the problem, that the quality of an ontology cannot easily be evaluated. This makes it hard to compare different approaches or to find the approach, that is best suited for a specific application.

The writing of the thesis turned out to be a quite time consuming task. This wasn't really unexpected, however, to write in a way, that it is scientifically and formally correct and to properly include relevant citations requires quite some effort.

7.2. Implementation and Testing

An important aspect of the implementation was, to carefully plan the interplay between the components of the system. As the prototype was developed using different programming languages for the different components, appropriate formats and interfaces for the data exchange were necessary. Therefore, data formats like XML or JSON came in handy, as there are libraries for all common programming languages available to process them.

Although Python was new to the author, it provided a good environment for the development of the NLP modules and to test the interface of the datastore and the according queries. Its advantages are, that it is easy to learn and intuitive and is also supported on all common platforms. The easy development of Python scripts proved to be especially useful for the implementation of algorithms which were newly developed, like for instance the NP clustering and taxonomy building, and therefore, have to be altered and adjusted repeatedly. However, for this kind of development, it has to be taken special care that the resulting module well fits into the whole system. Else it can become quite hard to maintain a proper structure for the framework.

What turned out to be more tricky than expected was to work with the Wikipedia API. Although it allows to automatically query all the available data, there are some issues that make it unnecessarily hard to get the desired data. The categorization system for instance contains content categories and maintenance categories, but doesn't provide an easy way to differ between these types. Furthermore, the Wikipedia data dump didn't contain all information it actually should have and was also not well documented and therefore wasn't useful at all.

The utilization of semantic web technologies for the datastore did not only work out well but was also a good way to get practice with these techniques. Thus, it was quite straightforward to establish the mapping from the example database to the general OLAP ontology, which is an important issue in the application of the proposed system.

The testing of the OLAP and Text Cube functionality gave some first practical insights into this technique. Especially the functionality that is enabled by the Text Cube operations opens up interesting new perspectives as it is a powerful extension to common IR techniques. This also motivates to further investigate how the potential of these new techniques can fully be utilized.

8. Conclusion and Future Work

The synergy of OLAP with text mining opens up some quite interesting new possibilities and challenges. An initial question is, how to store and organize the data, such that all potentially useful information can be accessed and how new information can be integrated.

For this purpose, the semantic web technologies turned out to be an appropriate choice, as they are designed in a way such that data of different sources can easily be merged and can therefore be made accessible in a homogeneous way. The available open source tools to set up RDF repositories and to develop ontologies were also very useful for the implementation of a prototype and experimenting with it.

To define a general schema for OLAP, such that the facts of a database can be described with the according dimensions, the general OLAP ontology and a mapping from a target database to that ontology was developed. This schema also provides the possibility to add additional dimensions at any time, as was demonstrated by integrating an extracted taxonomy. This can be done without the need to alter the original database and in principle for any kind of extracted information as long as it can be interpreted as an OLAP dimension.

Therefore, the taxonomy extraction can be seen as one possibility to add additional information, rather than the only way to extend the OLAP cube. However, the hierarchical structure of taxonomies suggest that it is a reasonable choice to try to use them as additional dimensions.

Although the single steps that were taken to extract taxonomies are relatively simple, the process delivered quite good results and still bears potential for improvements. A strength of this approach is, that it doesn't need preparation for a specific domain, or to define in advance of which domain the extracted taxonomies are. Furthermore, the only step that was performed manually, was to pick a cluster of NPs on which the relation finding process should be performed. This is one of the aspects that could still be improved. This could for instance be done by finding measures to rate the quality of a cluster and subsequently pick the best clusters automatically.

To use Wikipedia as background knowledge, turned out to well support the generality of the approach, as it contains many articles of very different domains. Nevertheless, using Wikipedia bears some technical difficulties. One problem is, that the API only allows one query per second, else the client IP address might get blocked. This restriction dramatically slows down the relation finding process, because as many terms are ambiguous, they require several queries to get all categories. Furthermore, categories in general have more than one parent category and therefore the number of necessary queries can become quite big. However, if the number of queries is kept as low as possible by querying only potentially useful NPs and not performing redundant queries, these issues can be handled quite well.

Thus, the proposed framework provides a reasonable way to implement database analyzing techniques for both, structured and unstructured data, which is also supported by the experiments with the implemented prototype.

Future Work

For the different parts of the system there are still a number of ideas that could be implemented and also several interesting questions are still open. The questions and extensions to the system, that seem most obvious, are discussed below.

Data Storage and Processing Speed

One open question is, how the single components scale for very large datasets and how updates of the dataset can be handled efficiently. For the datastore the question here is, how much data current implementations of RDF stores can handle and how fast they can process queries. As this technique is relatively new, compared for instance to relational databases, the implementations should probably not yet be expected to be technically mature. For instance, in the implementation it made a big difference if several simple queries or one more complex query was performed to fetch the same data. Here the simple queries turned out to be much faster. Thus, experiments with larger datasets would be useful to investigate such issues.

The OLAP engine could quite easily be improved, such that it can dynamically insert new datapoints, instead of importing the whole dataset at once from XML files. However, as currently the whole data is kept in main memory, the question is, if and how some data can be swapped to HDD if the dataset becomes too big to keep in memory, while still providing a short latency for OLAP operations.

Taxonomy Extraction

An important aspect of the taxonomy extraction is the quality of the underlying documents. If they come from online sources, which are likely to contain noise, text cleaning would be a useful preprocessing step, although, for large datasets, this can also become a computation intensive task.

Another aspect which would also be interesting for further investigation is how well the extracted taxonomies cover the dataset. For instance, if a number of very specific taxonomies is extracted, which cover only a small set of data points each, but together they cover most of the dataset, the domains of the taxonomies could be used for classification of the underlying documents or to create a kind of meta taxonomy. In both cases the result could be used as an additional dimension. Another possibility, to achieve more general taxonomies, is to adjust the NP selection such that NPs with a higher number of occurrences get a higher score. Then the resulting taxonomies should cover more facts as they would contain more general concepts.

The amount of background knowledge for the relation finding process could also be further increased by using additional sources, like for instance WordNet, OpenCyc or ConceptNet. This would replenish the available knowledge of Wikipedia and also compensate some of the issues discussed earlier.

Extraction of further Information

One of the greatest strengths of the framework is, that any kind of information, that can be interpreted as an OLAP dimension, can easily be integrated in the analyzing process. There are mainly two ways which would deliver additional information. The first one is to use another database that contains relevant information that can be related to the facts of the original database. The second way is to apply further data mining techniques to the available (text)data to extract additional information.

If several text mining techniques are applied, such that each yields another dimension, also a shift to a purely document centered analysis is thinkable. This would be especially useful if no or not enough structured information is available.

Nevertheless, it should be kept in mind, that there are some differences between a Text Cube and the traditional OLAP cube. As a document can have several values in a textual dimension, it does not necessarily belong to only one cell of the cube, but might be contained in several cells. Thus it might be sensible to investigate the impact of that issue on the analyzing process.

User Interface

An aspect that was beyond the scope of this work, but is very important nevertheless, is to provide a proper user interface such that the whole potential of the OLAP engine can be utilized. As the implemented OLAP engine provides many ways to instantiate a cube and to calculate different views on the dataset it is also important that an analyst can efficiently access all these possibilities.

Concluding it can be said, that the general architecture of the proposed system provides a good basis for a database analysis application that still provides much potential that awaits to be utilized in future work.

Bibliography

Abacha, A. B. and Zweigenbaum, P. (2011). A hybrid approach for the extraction of semantic relations from medline abstracts. In *CICLing (2)*, pages 139–150.

Abney, S. P. (1991). Parsing by chunks. In *Principle-Based Parsing*, pages 257–278. Kluwer Academic Publishers.

Banford, J., McDiarmid, A., and Irvine, J. (2010). Foaf: improving detected social network accuracy. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, Ubicomp '10, pages 393–394, New York, NY, USA. ACM.

Beckett, D. (2004). RDF/xml syntax specification (revised). W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

Bennett, M. (2006). Search 2.0 in the enterprise: Moving beyond "single shot" relevancy. *New Idea Engineering*.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Bock, J., Haase, P., Ji, Q., and Volz, R. (2008). Benchmarking OWL Reasoners. In *ARea2008 - Workshop on Advancing Reasoning on the Web: Scalability and Commonsense*.

Boussaid, O., Darmont, J., Bentayeb, F., and Loudcher, S. (2008). Warehousing complex data from the web. *Int. J. Web Eng. Technol.*, 4:408–433.

Brants, T. (2000). Tnt - a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing (ANLP-2000)*, pages 224–231. Association for Computational Linguistics.

Carroll, J. J. and Klyne, G. (2004). Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Carvalheira, L. C. C. and Gomi, E. S. (2007). A method for semi-automatic creation of ontologies based on texts. In *Proceedings of the 2007 conference on Advances in conceptual modeling: foundations and applications*, ER'07, pages 150–159, Berlin, Heidelberg. Springer-Verlag.

Chen, B., He, H., and Guo, J. (2008). Constructing maximum entropy language models for movie review subjectivity analysis. *Journal of Computer Science and Technology*, 23:231–239. 10.1007/s11390-008-9125-z.

Choi, Y., Kim, Y., and Myaeng, S.-H. (2009). Domain-specific sentiment analysis using contextual feature generation. In *Proceeding of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, TSA '09, pages 37–44, New York, NY, USA. ACM.

Cilibrasi, R. and Vitanyi, P. (2007). The google similarity distance. In *IEEE Transactions on Knowledge and Data Engineering*, pages 370–383.

Clough, P. (2001). A perl program for sentence splitting using rules. Technical report, University of Sheffield.

Codd, E., Codd, S., and C.T., S. (1993). Providing olap (on-line analytical processing) to user-analysts: An it mandate. Technical report, Codd & Date, Inc.

Dileo, J., Jacobs, T., and Deloach, S. (2002). Integrating ontologies into multiagent systems. In *Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002*, pages 15–16.

Ding, L. (2007). Provenance and search issues in rdf data warehouse. In *Proceedings of SemGrail 2007 Workshop*.

Fix, E. and Hodges, J. (1951). Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical report, Yale University, New Haven, CT USA.

Fodeh, S., Punch, B., and Tan, P.-N. (2011). On ontology-driven document clustering using core semantic features. *Knowledge and Information Systems*, pages 1–27. 10.1007/s10115-010-0370-4.

Giacometti, A., Marcel, P., Negre, E., and Soulet, A. (2009). Query recommendations for olap discovery driven analysis. In *Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pages 81–88, New York, NY, USA. ACM.

Giménez, J. and Márquez, L. (2004). Svmtool: A general pos tagger based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*.

Golbeck, J. and Rothstein, M. (2008). Linking social networks on the web with foaf: a semantic web case study. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, pages 1138–1143. AAAI Press.

Graña, J., Barcala, F.-M., and Ferro, J. V. (2002). Formal methods of tokenization for part-of-speech tagging. In *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '02, pages 240–249, London, UK, UK. Springer-Verlag.

Grefenstette, G. and Tapanainen, P. (1994). What is a word, what is a sentence? problems of tokenization. In *Third Conference on Computational Lexicography and Text Research (COMPLEX'94)*, pages 79–87.

Guha, R. V. and Brickley, D. (2004). RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Guo, H. (2007). *Automatic ontology extraction and applications*. PhD thesis, State University of New York at Stony Brook, Stony Brook, NY, USA. AAI3301487.

Habert, B., Adda, G., Adda-Decker, M., de Marëuil, P. B., Ferrari, S., Ferret, O., Illouz, G., and Paroubek., P. (1998). Towards tokenization evaluation. In Rubio, A., Gallardo, N., Castro, R., and Tejada, A., editors, *Proceedings First International Conference on Language Resources and Evaluation*, volume I, pages 427–431.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann, second edition edition.

Hearst, M. A. (1999). Untangling text data mining. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 3–10, Morristown, NJ, USA. Association for Computational Linguistics.

Horner, J., Song, I.-Y., and Chen, P. P. (2004). An analysis of additivity in olap systems. In *Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, DOLAP '04, pages 83–91, New York, NY, USA. ACM.

Hutchins, W. (1997). Looking back to 1952: the first mt conference. In *TMI-97: proceedings of the 7th International Conference on Theoretical and Methodological Issues in Machine Translation*.

Hutchins, W. (2004). The first public demonstration of machine translation: the georgetown-ibm system. In *AMTA-2004*.

International Data Corporation (2010). The digital universe decade - are you ready?

Janet, B. and Reddy, A. V. (2010). Article: Cube index: A text index model for retrieval and mining. *International Journal of Computer Applications*, 1(9):88–92. Published By Foundation of Computer Science.

Janet, B. and Reddy, A. V. (2011). Cube index for unstructured text analysis and mining. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*, ICCCS '11, pages 397–402, New York, NY, USA. ACM.

Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. In Nédellec, C. and Rouveirol, C., editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142. Springer Verlag, Heidelberg, DE.

Kaplan, R. M. (2005). A method for tokenizing text. In *Inquiries into Words, Constraints and Contexts*. CSLI Publications.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32:485–525.

Ko, Y. and Seo, J. (2009). Text classification from unlabeled documents with bootstrapping and feature projection techniques. *Inf. Process. Manage.*, 45:70–83.

Kozareva, Z. (2006). Bootstrapping named entity recognition with automatically generated gazetteer lists. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, EACL '06, pages 15–21, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kroeze, J. H., Matthee, M. C., and Bothma, T. J. D. (2003). Differentiating data- and text-mining terminology. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, SAICSIT '03, pages 93–101, , Republic of South Africa. South African Institute for Computer Scientists and Information Technologists.

Krötzsch, M., Patel-Schneider, P. F., Rudolph, S., Hitzler, P., and Parsia, B. (2009). OWL 2 web ontology language primer. W3C recommendation, W3C. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.

Kudo, T. and Matsumoto, Y. (2001). Chunking with support vector machines. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL '01, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lin, C. X., Ding, B., Han, J., Zhu, F., and Zhao, B. (2008). Text cube: Computing ir measures for multidimensional text database analysis. In *Proc. 2008 Int. Conf. on Data Mining (ICDM'08)*.

Majewski, P. and Szymanski, J. (2008). Text categorization with semantic commonsense knowledge: First results. In *Neural Information Processing*, pages 769–778. Springer Berlin / Heidelberg.

Malhotra, A. and Biron, P. V. (2004). XML schema part 2: Datatypes second edition. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

McGuinness, D. L. and van Harmelen, F. (2004). OWL web ontology language overview. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Merritt, K. (2002). Data warehousing and the internet: converging technologies. *J. of Internet Commerce*, 1:49–61.

Mikheev, A. (1999). Periods, capitalized words, etc. *Computational Linguistics*, 28:289–318.

O'Hare, N., Davy, M., Bermingham, A., Ferguson, P., Sheridan, P., Gurrin, C., and Smeaton, A. F. (2009). Topic-dependent sentiment analysis of financial blogs. In *Proceeding of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, TSA '09, pages 9–16, New York, NY, USA. ACM.

Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2:1–135.

Pedersen, T. and Jensen, C. (2001). Multidimensional database technology. *Computer*, 34(12):40 –46.

Pendse, N. (2007). The origins of today's olap products. *OLAP Report*.

Pendse, N. and Creeth, R. (1995). The olap report. Technical report, Business Intelligence.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

Ramshaw, L. A. and Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*.

Recio-Garcia, J. and Wiratunga, N. (2010). Taxonomic semantic indexing for textual case-based reasoning. In Bichindaritz, I. and Montani, S., editors, *Case-Based Reasoning. Research and Development*, volume 6176 of *Lecture Notes in Computer Science*, pages 302–316. Springer Berlin Heidelberg.

Rosenfeld, B. and Feldman, R. (2007). Clustering for unsupervised relation identification. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 411–418, New York, NY, USA. ACM.

Salles, T., Rocha, L., Pappa, G. L., Mourão, F., Meira, Jr., W., and Gonçalves, M. (2010). Temporally-aware algorithms for document classification. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 307–314, New York, NY, USA. ACM.

Sánchez, D. and Moreno, A. (2006). A methodology for knowledge acquisition from the web. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 10(6):453–475.

Sanchez, D. and Moreno, A. (2008). Learning non-taxonomic relationships from web documents for domain ontology construction. *Data & Knowledge Engineering*, 64(3):600–623.

Sarawagi, S., Agrawal, R., and Megiddo, N. (1998). Discovery-driven exploration of olap data cubes. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '98, pages 168–182, London, UK. Springer-Verlag.

Schmid, H. (2008). Tokenizing. In Lüdeling, A. and Kytö, M., editors, *Corpus Linguistics. An International Handbook*, Handbücher für Sprach- und Kommunikationswissenschaft/Handbooks of Linguistics and Communication Science. de Gruyter, Berlin, New York.

Shen, H. and Sarkar, A. (2005). Voting between multiple data representations for text chunking. In *Proceedings of the Eighteenth Meeting of the Canadian Society for Computational Intelligence*. Canadian AI.

Song, M., Lim, S., Kang, D., and Lee, S. (2006). Ontology-based automatic classification of web documents. In *Proceedings of the 2006 international conference on Intelligent computing: Part II*, ICIC'06, pages 690–700, Berlin, Heidelberg. Springer-Verlag.

Tang, J., Li, H., Cao, Y., and Tang, Z. (2005). Email data cleaning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 489–498, New York, NY, USA. ACM.

Tsuruoka, Y. and Tsujii, J. (2005). Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 467–474, Stroudsburg, PA, USA. Association for Computational Linguistics.

van Rijsbergen, A., Robertson, S., and Porteruthor, M. (1980). New models in probabilistic information retrieval. Technical report, British Library.

Whitelaw, C., Kehlenbeck, A., Petrovic, N., and Ungar, L. (2008). Web-scale named entity recognition. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 123–132, New York, NY, USA. ACM.

Witten, I. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition edition.

Wong, W. (2008). Discovering lightweight ontologies using the web. In *Proceedings of the 9th Postgraduate Electrical Engineering & Computing Symposium (PEECS)*, Perth, Australia.

Wong, W. (2009). *Learning Lightweight Ontologies from Text across Different Domains using the Web as Background Knowledge*. PhD thesis, University of Western Australia.

Wong, W., Liu, W., and Bennamoun, M. (2006). Integrated scoring for spelling error correction, abbreviation expansion and case restoration in dirty text. In *Proceedings of the fifth Australasian conference on Data mining and analystics - Volume 61*, AusDM '06, pages 83–89, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Wong, W., Liu, W., and Bennamoun, M. (2007a). Determining termhood for learning domain ontologies in a probabilistic framework. In *Proceedings of the 6th Australasian Conference on Data Mining (AusDM)*, Gold Coast, Australia.

Wong, W., Liu, W., and Bennamoun, M. (2007b). Determining termhood for learning domain ontologies using domain prevalence and tendency. In *Proceedings of the 6th Australasian Conference on Data Mining (AusDM)*, Gold Coast, Australia.

Wong, W., Liu, W., and Bennamoun, M. (2007c). Tree-traversing ant algorithm for term clustering based on featureless similarities. *Data Mining and Knowledge Discovery*, 15(3):349–381.

Wong, W., Liu, W., and Bennamoun, M. (2009). Acquiring semantic relations using the web for constructing lightweight ontologies. In *Proceedings of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, Bangkok, Thailand.

Yu, Y., Lin, C. X., Sun, Y., Chen, C., Han, J., Liao, B., Wu, T., Zhai, C., Zhang, D., and Zhao, B. (2009). inextcube: information network-enhanced text cube. *Proc. VLDB Endow.*, 2:1622–1625.

Zhang, D., Zhai, C., Han, J., Srivastava, A., and Oza, N. (2009). Topic modeling for olap on multidimensional text databases: topic cube and its applications. *Stat. Anal. Data Min.*, 2:378–395.

A. CD-ROM