Designing a rule-based annotation system to enhance semantic search on event-related articles

Master's Thesis

Jeremy Raes

Designing a rule-based annotation system to enhance semantic search on event-related articles

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Jeremy Raes
born in Antwerpen

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
http://www.wis.ewi.tudelft.nl/

Designing a rule-based annotation system to enhance semantic search on event-related articles

Author: Jeremy Raes
Student id: 1199404
Email: J.A.N.Raes@student.tudelft.nl

Abstract

Conventional search techniques are limited at solving complex queries because they work on the basis of word computations and link analysis. Semantic search promises to extend the paradigm of traditional IR from mere document retrieval to entity and knowledge retrieval by looking at the meaning of words. This MSc-thesis proposes a semantic annotation system capable of discovering various event-related entity types in articles on the web. The system will annotate these entity types with metadata linking them to instances stored in a semantic repository and link with their respective ontological class. Such annotation system can then further be used as a basis for developing a semantic search engine or other intelligent applications.

Thesis Committee:

Chair:	Prof.dr.ir. G.J.P.M. Houben, Faculty EEMCS, TUDelft
University supervisor:	Dr. F. Abel, Faculty EEMCS, TUDelft
Committee Member:	Dr.ir. A.J.H. Hidders, Faculty EEMCS, TUDelft
Committee Member:	Dr. A.E. Zaidman, Faculty EEMCS, TUDelft

Preface

The document before you is the conclusion of some great years studying here at the TU Delft. During this time I had the opportunity to travel the world and participate in various exciting projects, but I was equally inspired by the many people I've met and the different classes I took here on campus. Looking back, I am still dazzled by these wonderful opportunities and how they ended up shaping me – something I could never have expected when I first enrolled here at this university, now many, many years ago.

I would like to take this opportunity to thank the members of my thesis committee, starting with Geert-Jan Houben and Fabian Abel for their invaluable guidance and feedback. Jan Hidders and Andy Zaidman I would like to thank for agreeing to be part of my thesis committee. All of this is much appreciated. Raphael de Bois and Eric Feliksik I want to thank for helping me by proofreading this thesis document.

Even though not directly related to this Master's Thesis, I would like to take this opportunity to give a shout-out to all those people who have supported me throughout my thesis and my time here at the TU Delft: Both my parents, my brothers and sister for their love and support. JOBM, CDR and SatCM. Jan Raes, Olave Basabose, Menno Valkema, Freerk Wilbers, Henk Pijper, Willem Bult, Eva Hernando and Guido van Beek. Jenna Willis and the many other occupants of HB08.240. All of you made it even more worth while.

Jeremy Raes
Delft, the Netherlands
March 9, 2012

Executive summary

Problem definition

Looking on the web for suitable events and activities when visiting a certain region or just planning a night out, often proves to be a laborious task: the user does not specifically know what he is searching for and can at best define a set of boundary conditions based on his interests, time, location and budget. Current search engines see the texts in web pages as a sequence of characters and are thus ill-equipped for this type of queries. By annotating Web content with semantic descriptions one can solve such issues and create a semantic search engine. However, because annotation processes may require human efforts and can therefore be cost-intensive. In particular, for news-related and event-related content which may have a short lifespan these costs may exceed the utility and benefits that are gained by semantic annotations (such as more structured retrieval capabilities).

This MSc-thesis proposes a semantic annotation system capable of discovering various event-related entity types in articles on the web. The system will annotate these entity types with meta-data linking them to instances stored in a semantic repository and link with their respective ontological class. Such annotation system can then further be used as a basis for developing a search engine for the type of queries as described above.

Adding context to text: annotations and the semantic web

For machines to understand and collect information about the context, machine understandable semantic data needs to be added as annotations to the text. Knowledge representation languages such as RDF and the use of ontologies provide the necessary framework and form the foundation of what is coined the Semantic Web.

First, the different types of relationships between concepts in a certain domain should be defined in an ontology (e.g.: <book> <is_written_by> <author>). This ontology can then be used to reason about the different entities within that domain. Next, by defining explicit relationships between data (e.g.: <"Hollands Glorie"> <is_written_by> <"Jan

de Hartog">) machine-readable context can be added to what would otherwise just be flat text.

Because of the vast and ever growing collection of available documents on the web, it is virtually impossible to manually annotate documents. An automated annotation process provides the scalability needed to annotate existing documents, reduces the burden of annotating new documents, or allows for the use of multiple ontologies which can be beneficial to support the needs of different users.

Semantic annotation platforms (SAPs) can be classified based on the type of annotation method used. There are two primary categories: Pattern-based and Machine Learning-based. Pattern-based SAPs can either apply pattern discovery algorithms or have patterns manually defined. Machine learning-based SAPs utilize probabilistic methods or methods based on induction. Fully automated creation of semantic annotation remains however an unsolved problem, and all existing annotation systems will rely on human intervention at some point in the annotation process. The trade-off that one should make is that solutions based on machine-learning require an extensive training period, while developing a solution based on manually definined annotation rules is equally time consuming.

Architecture of a semantic search engine

High-level architecture

We propose a layered architecture for the annotation system and semantic search engine. The layers are built on top of each other with a clear division in functionality between them, so that layers can independently be added or removed, depending on changing requirements and use of different technologies. The layers are:

- The Data Acquisition Layer collects semi-structured (RSS, Atom) and unstructured pages (html).
- The Knowledge Acquisition Layer is responsible for the recognition of named entities in the collected texts and annotating them with links to their semantic class and description.
- The Data Layer provides a simplified access to the semantic and document repository.
- The Semantic Search Mechanism Layer groups the different mechanisms on which the semantic search services are built.
- The Semantic Search Services Layer groups the different functionalities a semantic search engine offers to the user.
- The Presentation Layer provides for the interaction between the user and the system.

The Data Acquisition Layer and The Knowledge Acquisition Layer form the annotation system, while the Semantic Search Mechanism Layer, Semantic Search Services Layer and the Presentation Layer are part of the actual search engine. The Data Layer is used by both sub-systems.

Implementation details of the annotating system

The implemented solution for the Knowledge Acquisition Layer that this research brought forth made use of GATE (General Architecture for Text Engineering), which is built around the pattern-matching grammar JAPE (Java Annotation Pattern Engine). This solution has the following advantages:

- Because GATE makes use of JAPE, which is a pattern-matching grammar, a training period is not required. This in contrast to pattern discovery algorithms or solutions based on machine learning.
- In GATE, different information can be added to an annotation. This functionality is known as a 'feature' and can be used to link a mention in a text to its proper ontological class (-es) and make it reference-able with an URI adding or linking it to its instance in the knowledge base.
- There are a multitude of different Tokenizers, Sentence splitters, POS-taggers, etc, available to GATE developers. In addition, it is also possible to custom develop language processing components for the GATE environment.
- GATE supports conditional pipelines that allow processing resources to be run or not according to the value of a feature on the document.

The language processing pipeline works as follows: First a tokenizer splits the text in very simple tokens. The second component, the gazetteer, loads the aliases of all the entity descriptions that are available to it via the semantic repository, and annotates those entities in the text for which it can find a matching alias. Next, a sentence splitter groups all tokens into sentences using regular expressions and based on that input, a part-of-speech (POS) tagger determines per token the correct word class.

The Named Entity Recognition (NER) component combines the information added by the gazetteer and the POS-tagger in a regular expression. This allows for new NE to be discovered. At this stage, newly discovered NE will only be annotated with a class identifier. An entity identifier, however, will be added by another component later.

The Annotation Cleaner component removes all temporary annotations created by the gazetteer and NER-component.

The Instance Generator component adds an identity identifier to the newly detected named entities that are not yet part of the semantic repository.

The Data Layer is responsible for storing all crawled documents and keeps a semantic repository containing the semantic descriptions of named entities.

Evaluation

In the evaluation we compare three different annotation strategies and evaluate how well they allow to annotating event-related content. The results of these different strategies are depicted in Table 1 and are briefly explained below.

Stategy	Precision		Recall		False Pos.	F– measure	
	Strict	Lenient	Strict	Lenient		Strict	Lenient
General Concept	0.527	0.74	0.509	0.589	0.057	0.453	0.628
Extended KB	0.711	0.876	0.768	0.806	0.011	0.711	0.797
Domain Grammar	0.818	0.887	0.914	0.915	0.007	0.850	0.895

Table 1: Evaluation metrics per strategy

For our first strategy, the General Concept strategy, which also serves as our baseline, we took a set of freely available pattern matching rules optimized to detect named entities in business articles from the Wall Street Journal, and a knowledge base of consisting of 80.000 instances. Even though this strategy is not intended to annotate event-related articles, it does fairly well at recognizing general concepts, but lacks sufficient knowledge to produce more precise results and is also responsible for a fair amount of false positives.

The Extended KB Stategy is built upon the General Concept strategy. For this strategy the most erroneous grammar rules where removed, and the knowledge base was extended with 30.000 domain specific instances derived from DBpedia. This strategy has the advantage that it can be implemented relatively fast, and was already able to significantly increase precision and recall compared the the baseline.

The third strategy, the Domain Grammar strategy, has access to the same instances in the knowledge base as the Extended KB strategy, and contains a set of custom built domain specific grammar rules. Challenge here is to create the right set of grammar rules – meaning, annotating as many meaningful concepts as possible while at the same time minimizing the number of spurious annotations. Combining the knowledge of the Extended KB strategy with the flexibility of grammar rules tailored for the specific domain, allowed us to significantly increase the number of named entities that we were able to recognize.

As expected, this strategy is overall the best equipped strategy to discover and annotate concepts. It can put forward strong results for both precision and recall, and only produces a few false positives.

Contents

Preface iii

Contents ix

List of Figures xi

I Background and context 1

1 Introduction 3
1.1 Background and motivation 3
1.2 Reseach goal 4
1.3 Document outline 5

2 The Semantic Web 7
2.1 Semantic Web Stack 8
2.2 Benefits of the Semantic Web 13
2.3 Conclusion 15

3 Annotating 17
3.1 Manual annotation approaches 17
3.2 Automated annotation approaches 18
3.3 Conclusion 21

II Design and architecture 25

4 Semantic Search Architecture 27
4.1 Annotation System 27
4.2 Data layer 39
4.3 Semantic search system 41
4.4 Conclusion 42

5 Evaluation 45

5.1 Methodology and metrics 45
5.2 Dataset 48
5.3 Annotation strategies 50
5.4 Results overview 51
5.5 Analysis of results 53
5.6 Synopis 58

III Conclusions and future work **61**

6 Conclusions and Future Work **63**
6.1 Conclusions 63
6.2 Future work 65

Bibliography **67**

List of Figures

1.1 Current search engines are unable to produce meaningful results to complex queries. 4

2.1 The Semantic Web Stack 9
2.2 An RDF graph representation of a person 11
2.3 An example of an ontology describing the location of a museum 13

3.1 Classification of annotation strategies 18
3.2 Flow chart of the DIPRE algorithm by Brin [12] 23
3.3 Markov process example 24

4.1 Architecture 28
4.2 JAPE Grammars 33
4.3 Ontology of the Event Annotation System 39

5.1 Compared to the annotations in the key set, a response annotation falls into one of these categories. 47
5.2 Example Key Set 49
5.3 Results Strict Evaluation 52
5.4 F-measure General Purpose Pipeline 52
5.5 F-measure KB 53
5.6 F-measure KB and Grammar rules 53

Part I

Background and context

Chapter 1

Introduction

1.1 Background and motivation

Over the past 20 years the the World Wide Web (WWW) has drastically changed the availability of electronically accessible information. Using a combination of keyword based and link navigation we can access more data than ever before. Despite this phenomenally success in terms of size and number of users, today's Web is fundamentally a relatively simple artefact and due to the enormous amount of data, it has become increasingly difficult to find, access, present, and maintain information required by a wide variety of users and applications. The reason to why machines find it difficult to process online data is twofold: information content is primarily presented in natural languages and making this data machine processable is a laborious process. Secondly, in the current hypertext paradigm a html anchor element is used to link two web pages. This element consists of a part of a web page as the 'anchor' of the link – this can be either text or an image – and an URI as the 'destination'. Human users are able to interpret the meaning of this link by examining its context, but this may not be so trivial for an automated process or "software agent".

The Semantic Web, envisioned by Tim Berners-Lee, is a means to an end for constructing a machine-understandable web of data [4]. The foundation of the Semantic Web is to add meaning or semantics to information that can be found on the World Wide Web as well as to make computers support humans performing otherwise laborious tasks by searching or annotating this data. This web of data can then be used for more advanced applications. For example: when one is looking for suitable events and activities when visiting a certain region or just planning a night out, it often proves to be a laborious task: the user does not specifically knows what he is searching for and can at best define a set of boundary conditions based on his interests, time, location and budget.

Because conventional search engines see text only as a sequence of characters, they are ill equipped for these types of queries and will at best generate a list of web pages containing a combination of the keywords that he entered. Unlike human users, they are unaware of the semantic relationships between concepts represented in web pages and cannot deal with synonyms (a word that means exactly or nearly the same as

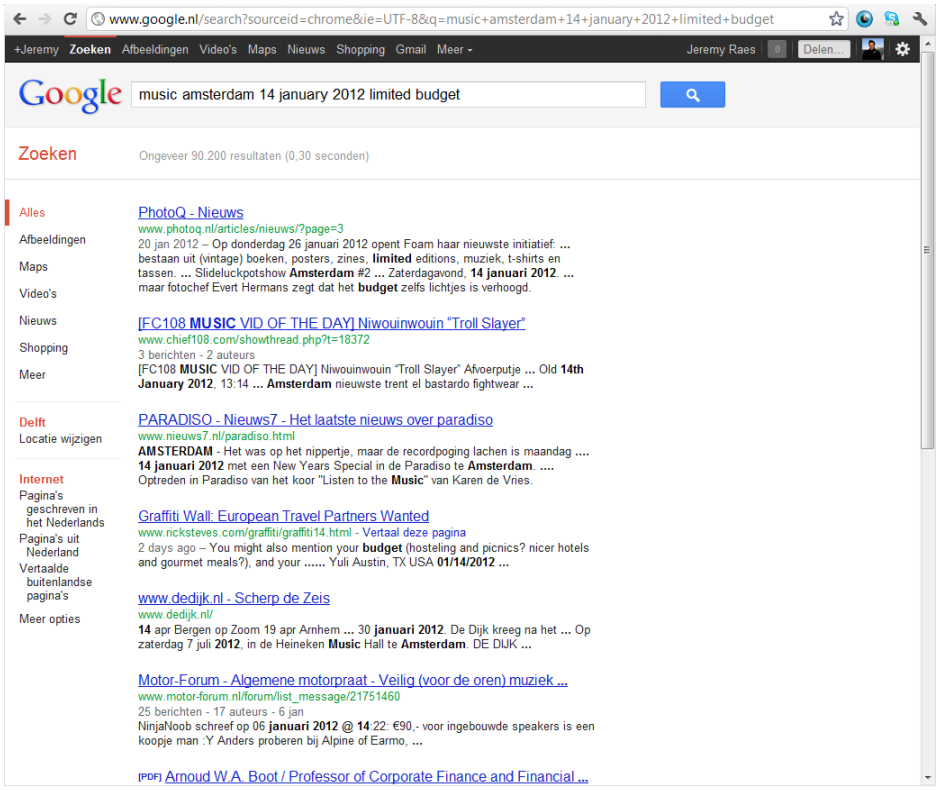

Figure 1.1: Current search engines are unable to produce meaningful results to complex queries.

another word), homographs (two words with the same spelling but a different meaning) and the inflections of words (a change in the form of a word, typically the ending, to express a grammatical function or attribute such as tense, mood, person, number, case, and gender). If, instead, the search engines would understand the semantics of the user boundary conditions set by the user (e.g.: “music”, “Amsterdam center”, “14/01/2012”, “limited budget”) and the web pages it crawls, it would become possible to present the user with more meaningful results.

A first step for developing such a system would be to annotate web pages with linked data. This MSc-thesis proposes a semantic annotation system capable of discovering various event-related entity types in articles. The system will annotate these entity types with meta data linking them to instances stored in a semantic repository and link them with their respective ontological class. In addition, this MSc-thesis will draft up an architecture for the overall search engine.

1.2 Reseach goal

Conventional search techniques are limited at solving complex queries because they work on the basis of word computations and link analysis. Semantic search promises to extend the paradigm of traditional IR from mere document retrieval to entity and

knowledge retrieval by looking at the meaning of words. This leads us to our main research goal:

> *Develop a strategy to semantically enrich and store event-related articles to allow for the retrieval of these articles by a domain specific search engine.*

In order to achieve this research goal, we will investigate the following research questions:

1. How can we discover and annotate entities in even-related articles?
2. How can we design a semantic search engine for event-related articles that incorporates functionality for automated semantic annotations and what are the different components of an underlying annotation system?
3. What is the quality of the different annotation strategies which are provided by the proposed system?

1.3 Document outline

This document is divided into three parts.

Part I identifies the background knowledge that is used throughout this document. Chapter 1, introduces the research topic and the research goal. Chapter 2 contains a brief description of the Semantic Web, the different technologies that support it and how they can be used to add meaning to text. Chapter 3 will give an insight in different strategies to add semantic annotations to documents.

Part II will answer the previously defined research questions. We start with Chapter 4, which comprises of an explanation our enrichment approach and all the different components that were used. Next, in Chapter 5, will evaluate this effort.

Part III concludes this document. Chapter 6 reflects on the research goal and research questions by discussing the main conclusions and contributions of this project. Also future work will be discussed in this chapter.

Chapter 2

The Semantic Web

The original World Wide Web was designed as an information space where documents are linked together using so-called hyperlinks [3]. These hyperlinks form a web of documents allowing for users to easily navigate from document to document.

The web pages themselves are marked up with languages such as HTML [47] and XML [2], which both convey some information about the structure of the document, but are mainly used as formatting languages. In combination with technologies as CSS [8], web designers are able to exactly tailor the look and feel of websites.

Because the before mentioned languages mainly deal with the lay-out of web pages, most of this information is solely designed for the purpose of human consumption. Unlike human users, computers see text only as a sequence of characters with no meaning, and therefore it is not evident for machines to understand the implications of the information captured inside these documents.

Likewise, because we humans are able to understand the text of hyperlinks and therefore we can easily understand how two linked documents are related to each other. A computer on the other hand can only see that a link between two web pages exists, but is unable to interpret why these two pages are linked together.

The objective of our efforts is to develop a system which is better equipped to deal with complex queries and therefore it is key to make the semantic context of event-related websites machine-accessible. This challenge can be tackled from two different angles: (1) either via an intelligent computer system which understand human languages, or (2) machine-readable information can be added to the data on the Web.

It can be argued that the first approach has been taken by Google[1], which analyses the structure of documents found on the Web, and the links between these documents, and infers information about the content, semantics and relevance of web pages based on this link structure [43]. However, the amount of computing power harnessed by Google [36] points to the high barrier to entry which exists in this approach.

The second approach allows the design of a system which is better suited to solving the problem of creating machine understandable content on the Web. It is with this

[1] http://www.google.com

in mind that the idea of the Semantic Web was conceived. The Semantic Web aims at making the semantic context of all data available on the web, machine accessible by adding structure to the meaningful content of web pages [4]. To do this, meta-data needs to be added to this meaningful content, describing it as a resource. This can be just about anything – such as documents, organisations, people, or other resources.

Once a resource is defined, it can be connected to other resources in a directed graph by means of the semantic relation that exists between them, forming a web of data. Machines can use this data to collect information about the current context of documents and use it to present the user with better search results.

To make this more concrete, consider the following example: the Semantic Web offers the possibility to define "John" as a *Person* and "Berlin" as a *City*. Next, it is possible to describe the (directed) relationship *lives in* between these two resources. Because it is a prerequisite for the Semantic Web that this information is machine-readable, a software agent is now, on the basis of this information, able to compile a list of all the persons living in Berlin.

After this short introduction , the remainder of this chapter is structured as follows: First we will explore different technologies that make up the Semantic Web Stack and how two core technologies, OWL and RDF, can be used to add meaning to text. After that we'll discus the advantages of the Semantic Web. We finish this chapter with a brief conclusion.

2.1 Semantic Web Stack

The Semantic Web Stack in Figure 2.1 is an illustration by Tim Berners-Lee, and depicts the hierarchy of languages used to construct the Semantic Web. Each layer exploits and uses capabilities of the layers below. The figure also shows how Semantic Web is an extension of classical hypertext web and not a replacement. The W3C[2] is responsible for developing and maintaining standards, which can all be retrieved at the W3C website[3].

Trust Trust will help verifying that the premises come from trusted source and that new information relies on formal logic. This is one of the unresolved issues in the Semantic Web Stack.

Proof The proof component within the Semantic Web is still under research. Its main objective is to be able to reason over (inferred) RDF data and determine whether parts of it can be proven correct.

Underlying Logic The underlying logic component is meant to mathematically unify the semantic worlds of RDF, Description Logics, and rules.

[2] W3C is short for the World Wide Web Consortium
[3] http://www.w3.org/

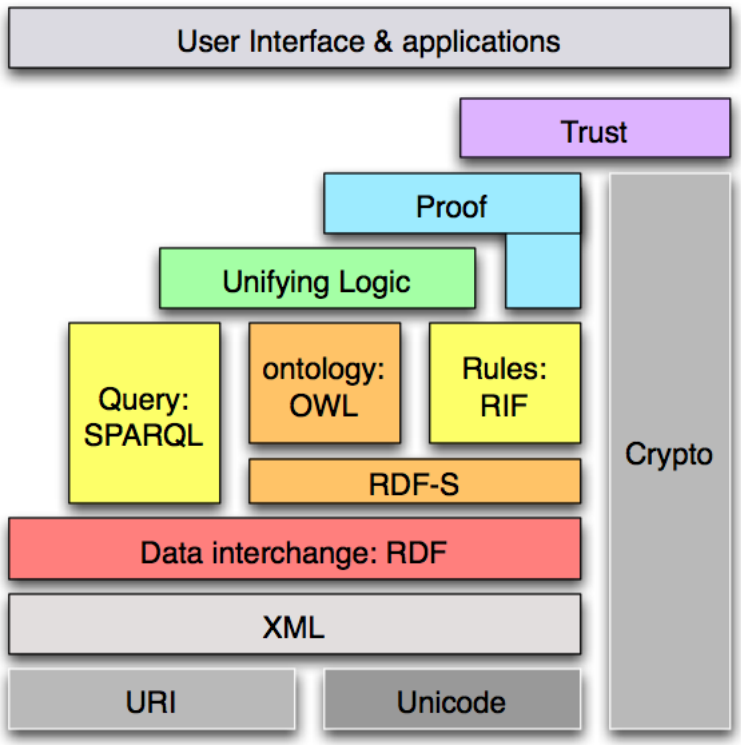

Figure 2.1: The Semantic Web Stack

(source: `http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/Overview.html#(1)`)

Cryptography Cryptography is important to ensure and verify that semantic web statements are coming from a trusted source. This is still an unresolved issue within the Semantic Web community.

SPARQL SPARQL Protocol and RDF Query Language (SPARQL) [45] is a query language for RDF and is considered as one of the key technologies of Semantic Web. A SPARQL-query is allowed to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

Ontology / OWL An ontology allows for the creation of a system of constraints based on the types of resources which are described by the data, and the properties that these resources have. The Ontology Web Language (OWL) [19] extends RDFS towards a more expressive ontology language, and is based on Description Logics (DLs) [41].

Rules / RIF The Rule Interchange Format (RIF) [31] is an effort towards the standardisation of a rules language for the Semantic Web. This is as well an unrealized technology in the Semantic Web Stack.

RDFS RDF Schema [10] is a vocabulary for describing properties and classes of RDF resources, with semantics for generalisation-hierarchies of such properties and classes.

RDF The Resource Description Framework (RDF) [33] is the basic data language for the Semantic Web. It is a framework for expressing relations between resources and data. RDF allows describing web-based resources by creating a directed and labelled graph model connecting resources (URIs) with other resources as well as literal data. RDF has a number of different representation formats, one of which is RDF/XML.

XML Extensible Markup Language (XML) [9] is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.

URI The Uniform Resource Identifier (URI) [39] is used to identify and locate resources, which can be regular hypertext pages on the World Wide Web. As mentioned previously, RDF uses HTTP URIs to describe data about a particular entity.

UNICODE Unicode is a standard for the consistent encoding, representation and the handling of text expressed in most of the world's writing systems.

Next we will focus on RDF and OWL as these are two key technologies from the Semantic Web Stack used in the proposed annotation system.

2.1.1 RDF

The Resource Description Framework (RDF) is a framework designed for describing and linking resources present on the World Wide Web and is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people [49]. At a conceptual level an RDF-model forms a directed graph of which the basic building is a subject-predicate-object triple: both the subject and objects are nodes in this graph, while the predicate is a directed arc connecting them [33]. The subject is the resource that is being described, the predicate denotes some property of the subject, and the object is the value of that property. This results in simple statements that are easy to understand and work with for both humans and machines. For example:

John (subject) lives in (predicate) Berlin (object).

The subject and predicate of an RDF-triple are usually and URI. Objects may be either a resource (a URI) or a literal value. Choosing an URI to name things has the advantage that they are unique and makes them reference-able on the Internet. In the example in Figure 2.2 there is a Person identified by `http://www.w3.org/People/EM/contact#me`,

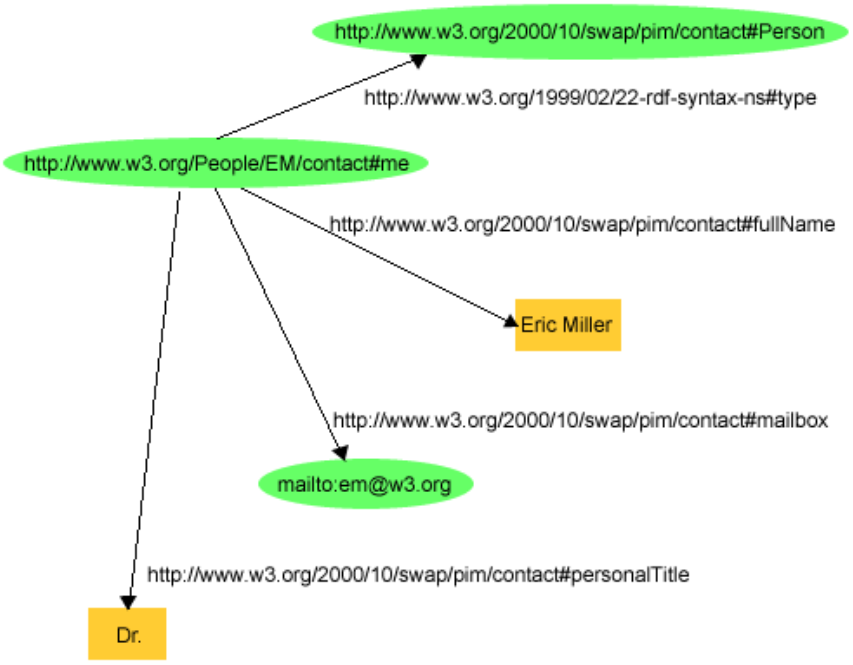

Figure 2.2: An RDF graph representation of a person

(source: `http://www.w3.org/TR/rdf-primer/`)

whose name is "Eric Miller" (literal value), whose email address is `em@w3.org`, and whose title is "Dr" (also a literal value).

RDF also provides a serialized XML-based syntax (called RDF/XML) for recording and exchanging these graphs. Each node in the RDF graph is described using an XML node, and each arc in the RDF graph is also described using an XML node. RDF graphs can be considered as a collection of paths which traverse the graph. Algorithm 2.1 shows an RDF/XML encoding of the RDF graph depicted in Figure 2.2.

Algorithm 2.1 An RDF graph encoded in RDF/XML

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
3   <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
4         <contact:fullName>Eric Miller</contact:fullName>
5         <contact:mailbox rdf:resource="mailto:em@w3.org"/>
6         <contact:personalTitle>Dr.</contact:personalTitle>
7   </contact:Person>
8 </rdf:RDF>
```

2.1.2 Ontology

Before multiple applications can truly understand data, and treat it as information rather than data, semantic interoperability is required. This can be done using an ontology. In the context of computer science we can define an ontology as a formal, explicit specification of a shared conceptualisation [25]. An ontology provides the shared vocabulary which can be used to model a domain, that is, the type of objects and/or concepts that exist, and their properties and relations. Important motivations for using are [42]:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

It is good practice to, wherever possible, reuse ontologies instead of creating new ones. In order to make it as easy as possible for machines to process your data, it is best to reuse terms from well-known ontologies. Only define new terms if it is impossible to find the required terms in existing ontologies, so that redundancy is minimised [13]. Within the sub-domain of Natural Language Processing (NLP) ontologies are mainly used to model both lexical and domain knowledge – domain knowledge which should be modelled in such a way that other software agents can easily make use of it. An upper ontology (top-level ontology, or foundation ontology) is an ontology which describes very general concepts that are the same across all knowledge domains[4]. This is usually a hierarchy of entities and associated rules (both theorems and regulations) that attempts to describe those general entities that do not necessarily belong to a specific problem domain. Upper Ontologies are quickly becoming a key technology for integrating heterogeneous knowledge coming from different sources. In fact, they may be exploited as a “lingua franca” by intelligent software agents [38].

OWL is a Web Ontology language [41]. It adds more vocabulary to describe properties and classes than RDF or RDF Schema: among other things, it can describe relations between classes (such as disjointness), cardinality (for example, "exactly one"), equality, richer typing of properties, and characteristics of properties (such as symmetry). OWL is designed for use by applications that need to process the content of information rather than just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema by providing additional vocabulary along with formal semantics. The basic components of an OWL ontology include classes, properties, and individuals. The difference is explained below:

[4] http://en.wikipedia.org/wiki/Upper_ontology_(information_science)

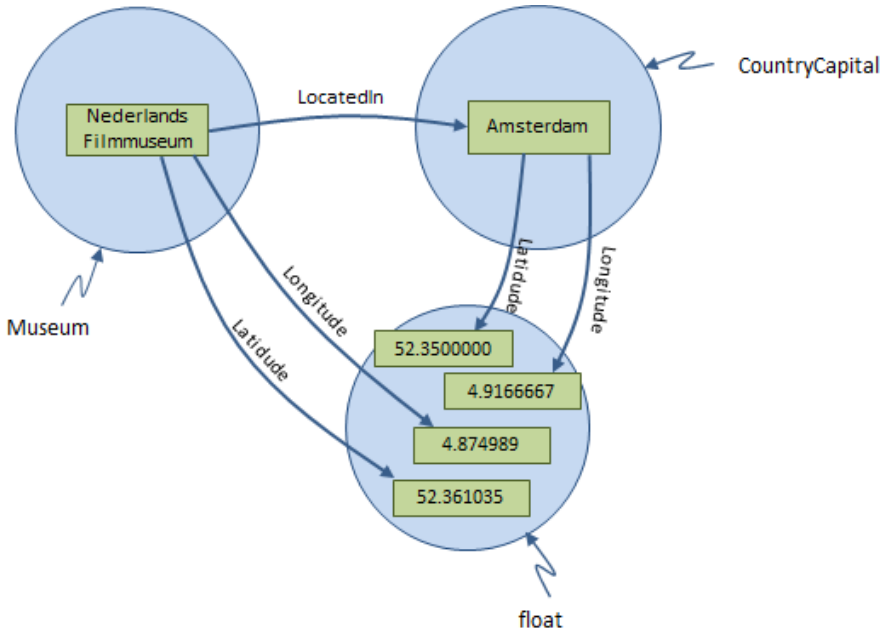

Figure 2.3: An example of an ontology describing the location of a museum

Classes Classes are the basic building blocks of an OWL ontology. They provide an abstraction mechanism for grouping similar resources within a domain based on their characteristics and usually constitute a taxonomic hierarchy (a subclass-superclass hierarchy).

Individuals Individuals, also called facts, are instances of classes, and properties can relate one individual to another. All the individuals that are members of a given OWL class are part of its class extension.

Properties OWL distinguishes between two main categories of properties: datatype properties and object properties. The first category relates individuals to datatype values such as integers, floats, and strings. The latter category relates individuals to other individuals.

In the example given in Figure 2.3 `Amsterdam` is considered as an individual of the class `CountryCapital`. The depicted `Latitude` property is an example of a datatype property, while `LocatedIn` is an obvious object property.

2.2 Benefits of the Semantic Web

Three key benefits which can be gained by creating a more machine-understandable web of knowledge are briefly discussed below:

1. Reuse and aggregation

2. Specific query and pattern analysis

3. Inference

Reuse and aggregation Technologies such as RSS [6], vCard and iCal have become popular machine readable formats for sharing contact information. Really Simple Syndication, or RSS, is a format to automatically publish frequently updated contents such as blog entries, news headlines, audio and video, in a standardised structure, while iCal and vCard are standards for sharing respectively calendar and contact information. Publicly accessible calendars can also provide re-usable (syndicated) content, which can be aggregated for personal or group use. Think for example about automatically updating your agenda with the right date just by downloading an iCal file after buying a ticket to a concert or booking a dentist appointment online. Likewise, this type of syndication allows service providers such as Bloglines[5], Technorati[6], and countless others, to reuse information from an RSS feed on their own web pages.

Geoblog[7] is another interesting example of the use of aggregated content whereby recently updated blogs are displayed on a world map, flashing titles of new posts as they are made. The idea of publishing information in a semantically rich format has been key to the recent explosion of blogs and aggregated news available in the RSS formats, thereby demonstrating the power of providing information in such rich formats.

Specific query and pattern analysis Currently, search engines operate on a free text analysis of documents intended for human use. This introduces the problem of ambiguity. For example, the search term "Rhodes" may refer to the island in Greece, the British politician Cecil John Rhodes, Rhodes College in Memphis Tennessee, or Rhodes University in Grahamstown South-Africa. It should be evident that any search engine attempting to provide a rudimentary search on this term will group all results, irregardless of the intended context. By offering a refinement based on the type of information the user is looking for, search engines may be able to offer better search results. Alternatively, a refinement in the form of a filter may be offered, for example filtering for web pages which relate to Rhodes University in Grahamstown, rather than Rhodes College in Memphis.

In understanding how different concepts interrelate, it is possible to understand more about a document than simply by looking at each concept in isolation. By creating a structured description of concepts and how they interrelate, the Semantic Web enables developers to analyse the structure of the relationships between concepts, which will lead to more powerful search tools or the possibility to find closely related concepts. This sort of pattern analysis can also allow improve search results, because a computer will be able to infer that two concepts, although labelled differently, and described by different resources (that is, located at different Uniform Resource Identifiers, or URIs) are the same thing because of the properties which they share. This seems similar to the topic of inference, but differs in that human (or other outside) specification of the links

[5] http://www.bloglines.com/
[6] http://technorati.com/
[7] http://www.brainoff.com/geoblog/

between concepts is not necessary. Whilst data-mining and text analysis techniques currently provide such results, the resulting descriptions are often "shallow" and often need to be enhanced by humans [23].

Inference By using well written vocabularies to describe information and a language which supports inference to express these descriptions, it becomes possible for a computer to infer information which was not expressed directly by a human when the information was created.

Lets take the example of someone selling his old iPhone at an online auction platform. When the user is entering the necessary information in the system, he is presented with the option of labelling the device being sold either as "wired telephone", "basic mobile phone" or "smart phone". Next, when publishing the information on the Web, the information would link to a vocabulary which describes telephones, and includes statements such as "all smart phones are mobile phones". A potential buyer, searching for all phones for sale, would because of this be able to infer that the iPhone for sale is also a mobile phone, and that item would therefore appear in the search results, even though the publishing website did not publish a statement saying that the item for sale was a mobile phone.

2.3 Conclusion

This chapter introduced the Semantic Web and demonstrated how it can be used to structure contextual information on resources by adding meta-data to text. These resources form nodes in a directed graph by means of the semantic relationship between them. Machines can use this data to collect information about the current context of documents and use it to present the user with better search results. We briefly introduced the different technologies that make up the Semantic Web Stack and then focused OWL and RDF – two key technologies for realising a semantic search engine: The primary purpose of an OWL ontology is to classify things in terms of semantics, or meaning. RDF is a language for describing and linking resources on the Web in a machine-readable format.

Chapter 3

Annotating

The previous chapter explained how the Semantic Web can benefit intelligent software agents to re-use and reason over information. The fulfilment of this semantic vision requires the widespread availability of semantic annotations for both existing and new documents on the Web [48].

Named Entity Recognition (NER) techniques can be used for processing texts and identifying certain occurrences of words or expressions belonging to a particular category. Once a named entity is recognised, it should be annotated. An annotation is a form of meta-data attached to a particular section of document content. This section may be a single word, a sentence or even a series of paragraphs. Every annotation should have a type (or a name) which is used to create classes of similar annotations, usually linked together by their semantics.

A semantic annotation distinguishes itself from other annotations in that respect that it formally identifies concepts and relations between concepts in documents, and links them to their respective semantic description. Compared with traditional NER approaches, semantic annotations are produced in an open and formal system, with clear semantics, and bound entity types. This encourages the re-use of both lexical resources and the resulting annotations by other systems [32].

There are two basic approaches for adding annotations to a text: automated and manual. Automatic annotation is less precise but can operate over many more documents than humans can reasonable address. Manual is to a point more precise, but very labour intensive and is therefore often only used for training purposes. Figure 3.1 [48] illustrates a classification of the different annotation strategies.

3.1 Manual annotation approaches

Manual annotation is easily achieved today, using integrated tools such as Semantic Word [50] and GATE Teamware [44]. The result of a human annotated text is often fraught with errors due to factors such as the annotator his familiarity with the domain, amount of training, personal motivation and complex schemas [1]. It is an expensive process, and often does not consider that multiple perspectives of a data source –

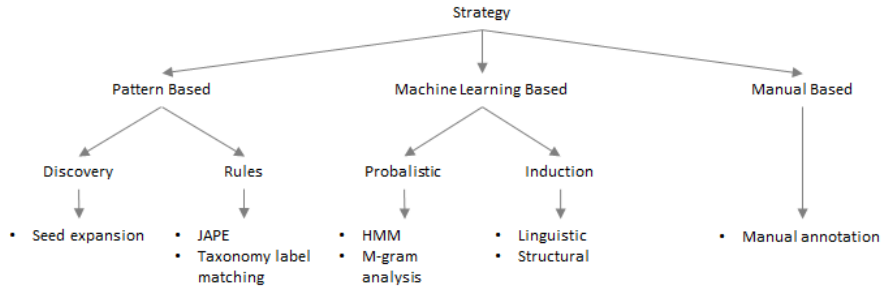

Figure 3.1: Classification of annotation strategies

requiring multiple ontologies – can be beneficial to support the needs of many users. Another problem with manual annotation is that documents and ontologies can change, requiring new or modified markup, which leads to document markup maintenance issues. Therefor, due to the volume of existing documents on the web that must be annotated to become a useful part of the semantic web, manual annotation has lead to a knowledge acquisition bottleneck [37].

To overcome the annotation acquisition bottleneck, (semi-) automatic annotation of documents has been proposed. The fully automated creation of semantic annotation remains an unsolved problem, and all existing annotation systems rely on human intervention at some point in the annotation process. Manual annotation systems are therefore often only used to train a machine to perform automatic annotation, evaluation and quality assurance, or on one small and specialised corpora, whilst automated annotation approaches provides the scalability needed to annotate existing documents on the web, and reduces the burden of annotating new documents.

3.2 Automated annotation approaches

Because of the vast and ever growing collection of available documents on the web, it is virtually impossible to manually annotate documents. An automated annotation process provides the scalability needed to annotate existing documents, reduces the burden of annotating new documents, or allows for the use of multiple ontologies which can be beneficial to support the needs of different users. In addition, automated approaches provide scalability by reducing or potentially eliminating the human workload.

Semantic annotation platforms (SAPs) can be classified based on the type of annotation method used. There are two primary categories: Pattern-based and Machine Learning-based. Pattern-based SAPs can either apply pattern discovery algorithms or have patterns manually defined. Machine learning-based SAPs utilise two methods: probability and induction. Probabilistic SAPs use statistical models to predict the locations of entities within text. Most automated annotation approaches however use a multistrategy, meaning that they use a combination of techniques.

3.2.1 Pattern matching based annotation processes

Discovery-based strategies

Most pattern-discovery methods follow the Dual Iterative Pattern Relation Expansion (DIPRE) outlined by Sergey Brin [12]. An initial set of entities is defined and the corpus is scanned to find patterns in which similar entities exist. In this process new entities are discovered along with new patterns. This process is repeated recursively until no more entities are discovered or the discovery process is halted. Figure 3.2 demonstrates this algorithm in more detail using a flow chart.

Ideally the pattern is specific enough not to match any tuples that should not be in the relation, however, in practise a few false positives may occur. Patterns may have various representations, e.g.: regular expressions.

Rule-based strategies

There are two strategies for injecting knowledge using rule-based pattern matching methods, which we'll briefly discuss below.

Dictionaries

An easy way to find occurrences of named entities in a text and annotate them accordingly, is to keep a list of possible NE and look for matches in texts you'd like to annotate. In its simplest form, a dictionary consists of a set of lists containing names of entities and the class to which they belong. To annotate texts the only thing that needs to be done, is to compare all the words (or tokens) in the text with the named entities defined in the dictionary and see if a match can be found. This relatively simple concept has of course been further developed, and a wide range of different types of gazetteers are now available to the NLP-engineer.

Even though dictionaries are very time-efficient at run-time, it often proves to be a time consuming task to create and maintain them. Furthermore, they have a low precision (e.g.: "Washington" can either be a state, city, or a person), and their recall decreases as the number of instances grow.

Because we're most interested in creating semantic annotations, that is, an annotation connecting an entity to a semantic database or ontology, it would be interesting to use the instances stored in a semantic repository as a source for our gazetteer instead of flat text files. Such a solution would have the advantage that it can re-use named entities discovered by other other methods and that the annotations already refer to both the correct instance and class URI.

Java Annotation Pattern Engine (JAPE)

JAPE [18] is a component of the open-source platform GATE (General Architecture for Text Engineering) and provides finite state transduction over annotations based on regular expressions. Each JAPE grammar consists of a set of phases, which in turn consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. The left-hand-side (LHS) of the

rules consist of an annotation pattern description that may contain regular expression operators (e.g. +, *, ?, +). The right-hand-side (RHS) consists of the annotation manipulation statement. Annotations matched on the LHS of a rule may be referred to on RHS by means of labels that are attached to pattern elements.

A JAPE grammar is usually part of a language processing pipeline, and is able to use the output of the preceding components or other grammar rules to determine a match. The most important, but not limited to, of the other components in the language processing pipeline are:

Tokenizer A tokenizer splits the text into very simple tokens, such as numbers, punctuation and words of different types.

Gazetteer A gazetteer identifies key words in the text.

Part-Of-Speech Tagger A POS tagger adds the right word class to each token.

Rules based systems typically contain hand-crafted rules that define how entities can be found in text. Even though this is a time consuming task, it enables the NLP-engineer to tailor the rules to the domain in which they will be applied.

3.2.2 Machine learning based annotation process

Induction strategies

Algorithms based on induction automatically generate rules for Natural Language Processing based on sets of conjunctive conditions of adjacent words. To demonstrate this principle we will use the $(LP)^2$ algorithm as explained in [15]:

$(LP)^2$ learns from a training corpus where a user has highlighted the information to be extracted with different tags. These user generated tags are considered positive examples. The rest of the corpus is considered a pool of negative examples.

For each positive example the algorithm: (1) builds an initial rule, (2) generalizes the rule and (3) keeps the k best generalizations of the initial rule. In particular $(LP)^2$'s main loop starts by selecting a tag in the training corpus and extracting from the text a window of w words to the left and w words to the right. Each information stored in the $2*w$ word window is transformed into a condition in the initial rule pattern, e.g. if the third word in the window is "seminar", the condition on the third word in the pattern will be word="seminar". Each initial rule is then generalized.

In the generalization process $(LP)^2$ uses generic shallow knowledge about Natural Language as provided by a morphological analyzer, a part-of-speech tagger (indicating the word class for each word) and a user defined gazetteer. A lexical item (LexIt in the following) summarizes such knowledge for each word (e.g., companies) via: a lemma (company), a lexical category (noun), case information (lowercase) and a list of user defined classes as defined by a user-defined gazetteer (if available).

Generalization consists in the production of a set of rules derived by relaxing constraints in the initial rule pattern. Conditions are relaxed both by reducing the pattern

in length and by substituting constraints on words with constraints on some parts of the additional knowledge.

The last step of the algorithm is the selection of the best generalizations. Each generalization is tested on the training corpus and an accuracy score $L = wrong/matched$ is calculated. For each initial instance the k best generalizations are kept that: (1) report better accuracy; (2) cover more positive examples; (3) cover different parts of input 1; (4) have an error rate that is less than a specified threshold.

The other generalizations are discarded. Retained rules become part of the best rules pool. When a rule enters the best rules pool, all the instances covered by the rule are removed from the positive examples pool, i.e. covered instances will no longer be used for rule induction. Rule induction continues by selecting new instances and learning rules until the pool of positive examples is void.

Probalistic strategies

The Hidden Markov Model (HMM) is a popular statistical tool for modelling a wide range of time series data. HMMs are still a novel approach in the context of natural language processing, but have nonetheless already been applied with great success to problems such as part-of-speech tagging and noun-phrase chunking [5].

A Hidden Markov Model is based on the Markov assumption which says that the probability of a particular thing appearing in a sequence depends only on the one category before it, in other words working with bigram probabilities is good enough. Underlying each HMM is set of hidden (unobserved) states in which the model can be (e.g. parts of speech), with probabilistic transitions between states over time. Figure 3.3 shows an example of Markov process for a simple grammar and how we can move to a different state with a finite probability.

A problem with HMMs is that they require a set of data to train the model with. Once the states of the model have been defined, the transition probabilities and the output probabilities associated with each state need to be trained. Training normally occurs by setting the transition and output probability matrix values to some arbitrary starting value, and then iteratively refining the probability values in each matrix using the training data [22]. There are two standard approaches to this task: if the training examples contain both the inputs and outputs of a process, we can perform supervised training by equating inputs to observations, and outputs to states. If only the inputs are provided in the training data then we must used unsupervised training to guess a model that may have produced those observations.

See [46] for an extensive introduction on the application of HMM in natural language processing.

3.3 Conclusion

Most semantic annotation platforms for the Web have only recently been developed and it's not trivial to come up with dimensions to compare these different strategies. To

the best of our knowledge, there exists no study that compares these different strategies in terms of, for example, performance (precision and recall), training time, or how hard it is to implement a cetrain strategy. Other dimensions to distinguish annotation platforms by are extraction method use to find entities within a document, machine-learning (ML), if the platform allows for manual rules (MR), and wheter or not the platform is extensible.

Strategy	ML	MR	Platform(s)
Patterery discovery	No	Yes	Armadillo [21]
		No	Ont-O-Mat: PANKOW [14]
Probalistic	Yes	No	AutoBib [24], DATAMOLD [7]
Rule-based	No	Yes	AeroDAML [34], GATE[26], MUSE [40]
		No	MnM [51], SemTag [20]
Induction	Yes	No	Ont-O-Mat: Amilcare [27]

Table 3.1: Comparison of the different annotation strategies and platforms

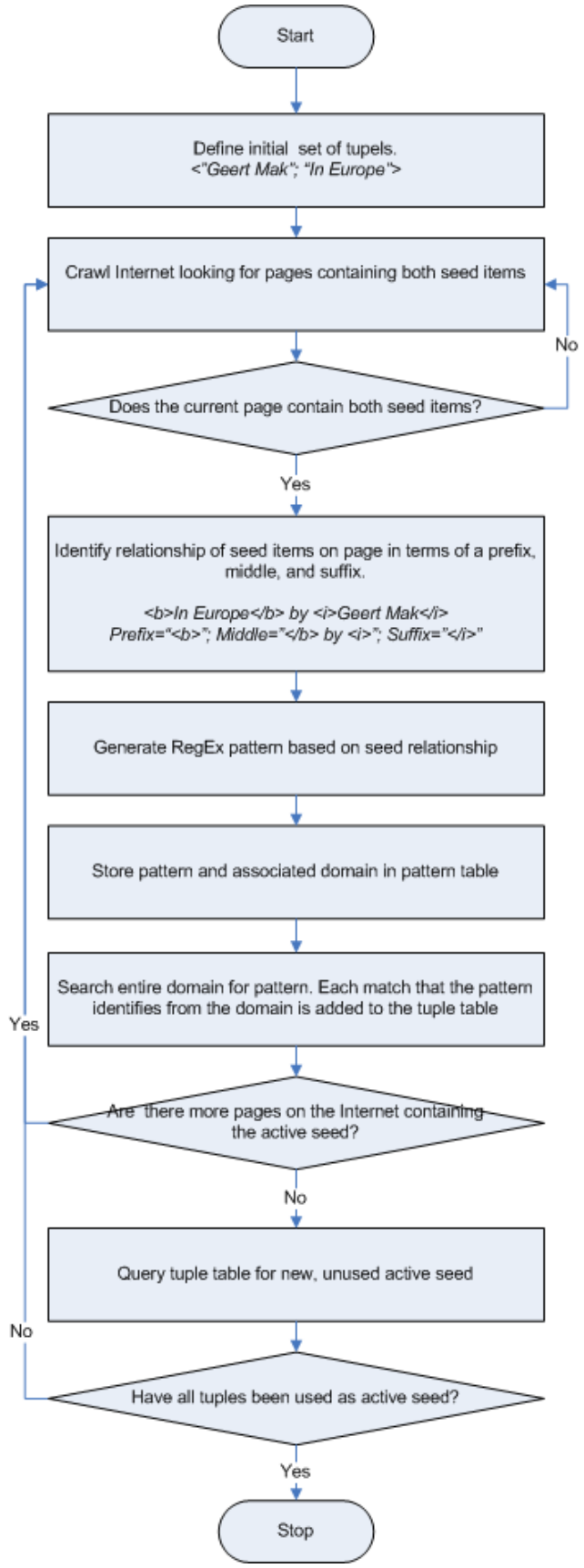

Figure 3.2: Flow chart of the DIPRE algorithm by Brin [12]

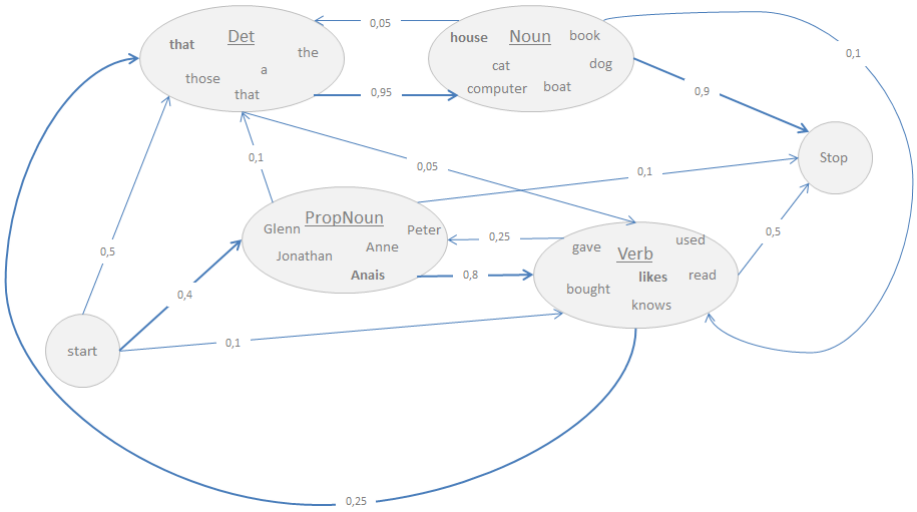

Figure 3.3: Markov process example

Part II

Design and architecture

Chapter 4

Semantic Search Architecture

This chapter presents a holistic architecture for an annotation system and semantic search engine, consisting of six distinctive layers: semantic data acquisition, knowledge acquisition, data integration, semantic search mechanism, semantic search services, and result presentation layer (see also Figure 4.1). Each layer unambiguously shields off implementation details, defines a set of functionalities, and defines a required input and an expected outcome. The combination of these layers form two logical systems: the first pair layers form a semantic annotation system, while the combination of the last three layers form a semantic search engine. A data layer for storing and retrieving documents and their respective annotations, plus a semantic repository containing a large collection of instances, is shared between these two logical systems.

4.1 Annotation System

Key decision in the development the annotation system is the choice for a specific annotation platform. Different annotation platforms and strategies were outlined in chapter 3. While each of these individual solutions have their own strengths and weaknesses, it was at the time of implementation unclear which solution would be the most beneficial in terms of precision, recall, or development time. Therefore was opted to choose for the most extensible platform. GATE[1] is an open source platform developed by the university of Sheffield and comes with a customisable and extendable set of components. It integrates well with third party solutions and is actively used in language engineering research.

4.1.1 Data acquisition layer

The data acquisition component collects semi-structured (e.g. RSS and Atom-feeds) and unstructured pages (html). Some links will contain all data on the same page, while other pages will also need to have their respective links examined.

[1] GATE is short for General Architecture for Text Engineering

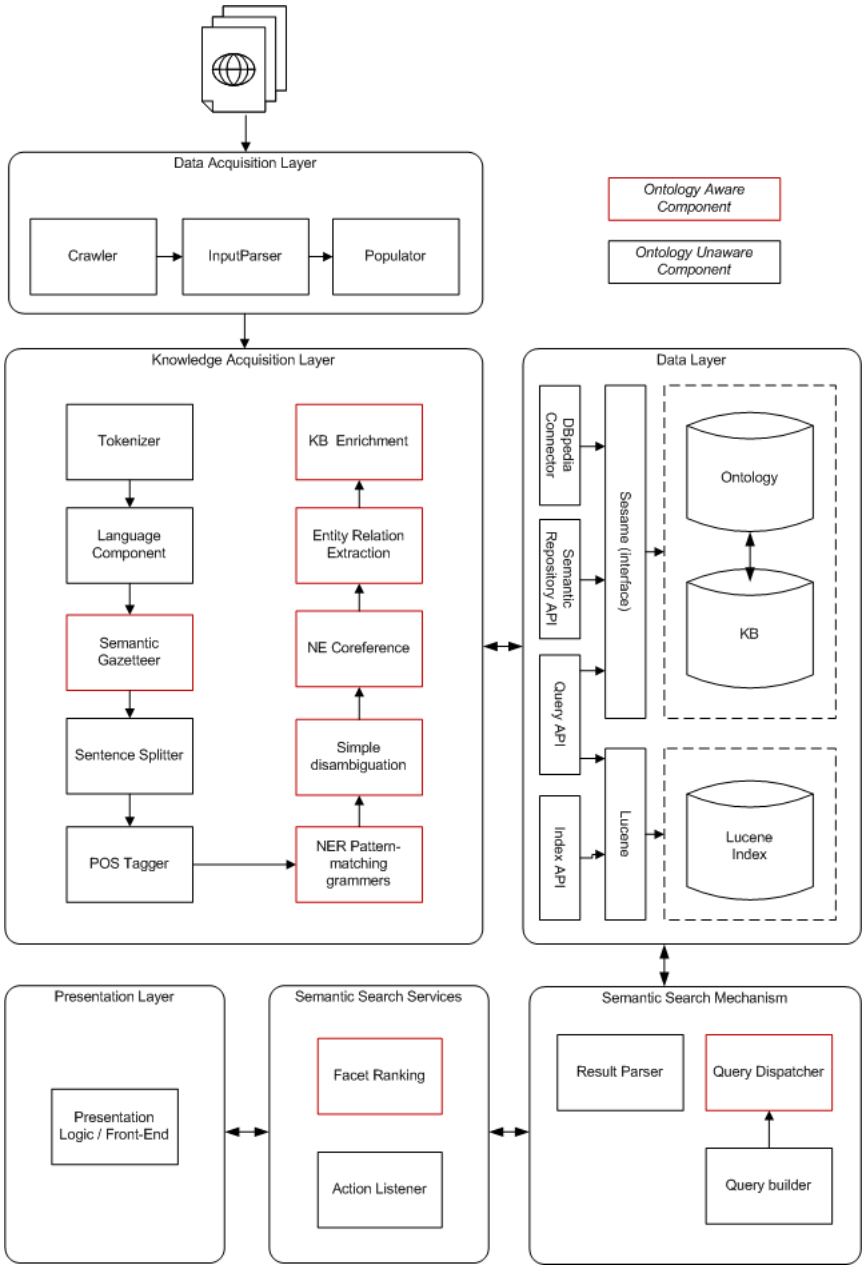

Figure 4.1: Architecture

Crawler

The crawler holds a list of URL's from which it harvest data. The crawler iterates through this list and determines per URL the content type, in order that the right type of InputParser can get selected.

InputParser

The InputParser receives an URI to download from the Crawler. This can be, for example, a web page or a RSS-feed. The InputParser first examines the data and, based on that, decides what to do with it. For example, in case of an RSS or Atom-feed the dedicated InputParser will first add every single item to the Populator. Next, the populator will send each RSS-item first to the the Knowledge Acquisition layer, and send then it annotated back to the InputParser. Items containing little or no relevant annotations are discarded while relevant items are send to the Data Layer for storage. If, for example, the InputParser is offered a link aggregation site containing various links to web pages describing an event, the InputParser will collect all these links and add them to the Crawler's URL list.

Populator

The populator takes the data from the InputParser and feeds it to the Knowledge Acquisition Layer so that the text can be annotated. When annotated, both the text and a separate file containing the annotations are send back to the InputParser, which will determine what to do next.

4.1.2 Knowledge acquisition layer

The collected unstructured and semi-structured data can be transformed to structured data using Natural Language Processing (NLP). The essence of the Knowledge Acquisition Layer is the recognition of named entities, and linking them to their semantic description in the Knowledge Base. This with respect to the entity classes defined in an ontology. Components in GATE form a conditional pipeline were decisions can be taken to select certain components, depending on the output of previous components. There are a multitude of different tokenizers, sentence splitters, POS-taggers, etc, available to GATE developers, but it is also possible to custom develop language processing component for the GATE environment. Withing GATE, JAPE transducers are used to manipulate annotations on text.

Tokenizer

In linguistics, tokenization is the process of breaking a sequence of characters into words, numbers, punctuations, and other symbols. These words and expression sequences are called tokens, and the tools performing such tokenization are called tokenizers. Various tokenizers are available as open source software and produce different

results. For example, some tokenizers split hyphenated compound words (e.g.: code-snippet) into two or more tokens (“code” “snippet”), but other tokenizers keep it as one token (“code-snippet”). Even though these type of distinctions might seem trivial at first, they may considerably alter the course of other natural language processing procedures such as indexing and part-of-speech (POS) tagging.

The aim is to limit the work of the tokeniser to maximise efficiency, and enable greater flexibility by placing the burden on the grammar rules later on, which are much more adaptable. To the best of our knowledge there was only very limited research available comparing different Tokenizers. Because of this we started off with GATE’s standard English Tokenizer (ANNIE).

Every token gets annotated, specifying its kind (word, number, space, punctuation) and, in case of ’words’, its orthographic classification (lowerCaps, uperCaps, mixedCaps, upperInitial). The following code-snippet is a sample output of the Tokenizer analysing the sentence “800,000 US dollars”:

```
Token, string = '800', kind = number, length = 3
Token, string = ',', kind = punctuation, length = 1
Token, string = '000', kind = number, length = 3
SpaceToken, string = ' ', kind = space, length = 1
Token, string = 'US', kind = word, length = 2, orth = allCaps
SpaceToken, string = ' ', kind = space, length = 1
Token, string = 'dollars', kind = word, length = 7, orth = lowercase
```

Language component

Based on the language recognised by the language component, the right components can be selected for the pipeline. In the current implemented NLP-pipeline the language component is used to disregard of all non-English texts.

Gazetteer

The role of the gazetteer is to identify named entities in the text based on lists. This project made use of two different types of implementations.

Large KB Gazetteer

This component originated as a part of the KIM platform but is now freely available in most versions of GATE. It provides a scalable implementation of vocabulary storage and fast gazetteer lookup and allows for loading vocabularies by querying both RDF files and semantic repositories.

The large KB gazetteer provides support for ontology aware NLP. It identifies all mentions in texts based on already existing named entities and Lexical Resources in the Knowledge Base and annotates them with two URIs:

- Named Entity URI: This is an URI that links a named entity in a text to its semantic representation in the Knowledge Base. All named entities have a unique instance URI, but can have several labels attached to it.

- A class URI: The ontological class to which the named entity belong to.

In order to prevent the system from repeating mistakes, we've set the query so that only entities that are marked 'trusted' in the KB are used by the gazetteer in the recognition process. The labels used by the gazetteer should meet the following three requirements:

- have at least one alias (label);
- are of type that is subclass of protons:Entity;
- are marked as Trusted;

Potentially there are multiple entity-aliases in the KB that are equivalent to named entities mentioned in the text and at this stage it is not yet possible to distinguish between entities that happened to share an alias. Therefore, at this stage, all equivalent possibilities are used to annotate the text and it is only in a later stage that disambiguation techniques are applied.

ANNIE Gazetteer

Because the (non-opensource) Large KB Gazetteer is unable to annotate sub-parts of tokens, a second gazetteer was added to the NLP-pipeline. The ANNIE gazetteer is a lightweight component part of the standard GATE configuration and works with plain text files as dictionary lists. We added two lists to recognise lexical resources of the following categories: PublicSquare (e.g.: "straat", "plein") and Theatre.

Unlike the Large KB Gazetteer, the standard ANNIE is not ontology aware and is only able to annotate (sub-) tokens with a so called major and minor type. This however was not seen as a major obstacle since the annotation of lexical resources is only required for the discovery of new named entities, and are not part of the definitive set of annotations.

Sentence splitter

A RegEx sentence splitter is used. As its name suggests, the RegEx splitter is based on regular expressions. This sentence splitter was chosen over the standard ANNIE splitter because of reported performance issues. The Java-based RegEx sentence splitter also allows to define custom regular expression patterns, but this proved not to be necessary.

Part-of-speech tagger

Part-of-speech (POS) tagging, also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition, as well as its context — that is to say the relationship with adjacent and related words in a phrase, sentence, or paragraph. The ANNIE POS tagger is based on the well-known rule-based Brill tagger [11].

Every Token is annotated with a word class after the tagging process. Some examples of the in total 54 different word classes are:

- CC - coordinating conjunction: 'and', 'but', 'nor', 'or', 'yet', plus, minus. less, times (multiplication), over (division). Also 'for' (because) and 'so' (i.e., 'so that').
- DT - determiner: Articles including 'a', 'an', 'every', 'no', 'the', 'another', 'any', 'some', 'those'.
- VB - verb - base form: subsumes imperatives, infinitives and subjunctives.
- MD - modal: All verbs that don't take an '-s' ending in the third person singular present: 'can', 'could', 'dare', 'may', 'might', 'must', 'ought', 'shall', 'should', 'will', 'would'.
- NNP - proper noun - singular: All words in names usually are capitalised but titles might not be.
- ...

Named Entity Recognition (NER) Pattern-matching grammars

Hitherto we've annotated the text with lexical resources, previously already known named entities, and, for each individual token, its respected word class. Using pattern matching grammars we combine this knowledge and use it to discover new named entities in texts. The challenge is to create the right set of grammar rules, and annotate as many meaningful concepts as possible while at the same time minimising the number of spurious annotations.

JAPE is a Java Annotation Patterns Engine. JAPE provides finite state transduction over annotations based on regular expressions. A JAPE grammar consists of a set of phases, each of which consists of a set of pattern/action rules. Each grammar rule always has two sides: left and right. The Left Hand Side (LHS) of the rule contains the identified annotation pattern that may contain regular expression operators (e.g. *, ?, +). The Right Hand Side (RHS) outlines the action to be taken on the detected pattern and consists of annotation manipulation statements. Annotations matched on the LHS of a rule are referred on the RHS by means of labels, for example:

LHS (regular expression for annotation pattern):

> *i.e., Look for a combination of FirstName annotation added by the gazetteer followed by a Token starting with a capital and label it Person.*

RHS (manipulation of the annotation patter from LHS):

> *i.e., Get the gender of the FirstName, if gender=male re-label Person as Male, else re-label as Female.*

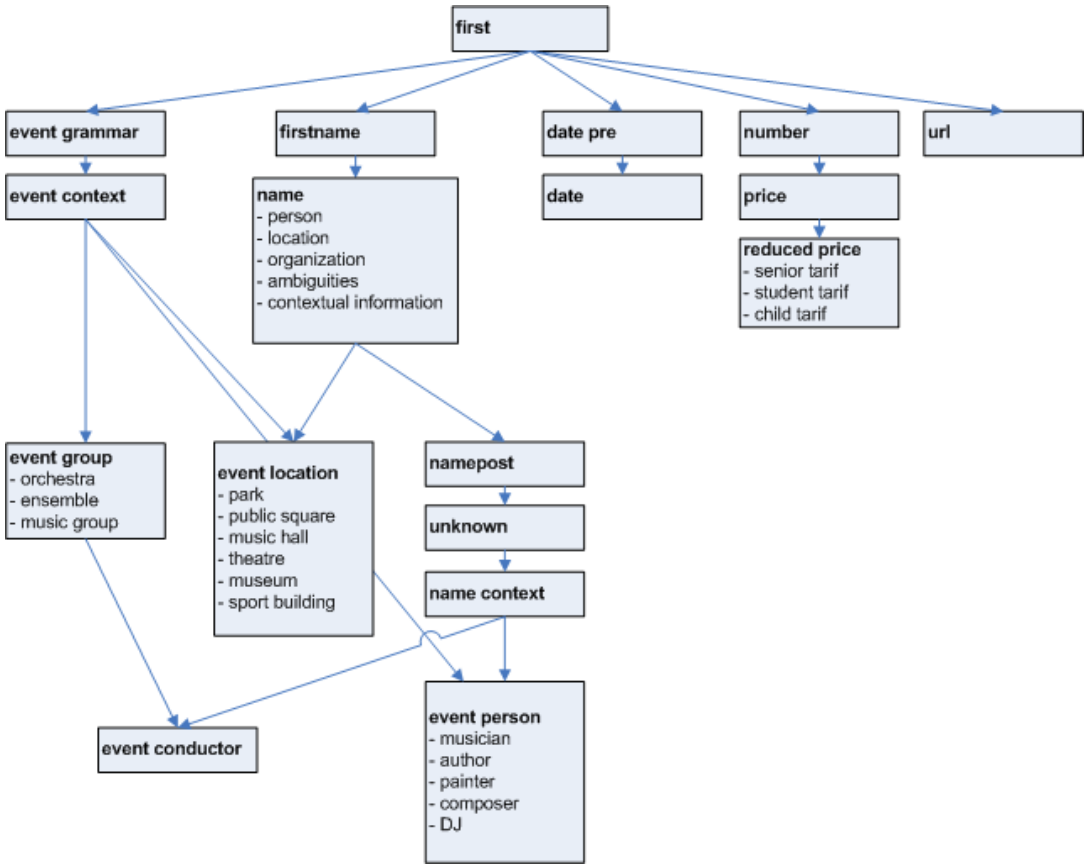

Figure 4.2: JAPE Grammars

We used a modified version of JAPE, making it possible to use the ontology in its evaluation.

Grammars:

Next we will describe the individual grammars used in the language processing pipeline and how they are combined together. While these grammars always run in fixed sequential order, not all grammars are dependent on each other. The grammars firstname, name, unknown, name context, date pre, date, url, and number are part of GATE [17]. Because these rules were originally written for a corpus of business articles taken from the Wall Street Journal, they were altered where required. The dependencies between grammars are depicted in Figure 4.2.

main This file contains a list in the correct processing order of all the grammars described below.

first This grammar contains a set of general macros used by the whole grammar set.

firstname This grammar contains rules to identify first names and titles via the gazetteer lists. It adds a gender feature where appropriate from the gazetteer list. This gender feature is used later in order to improve co-reference between names and pronouns. The grammar creates separate annotations of type FirstPerson and Title.

name This grammar contains an initial set of phases and rules to detect location and person entities. These rules all create temporary annotations, some of which will be discarded later, but the majority of which will be converted into final annotations in later grammars. Some rules are negative rules. These types are mainly used to add context information to prevent other annotation rules from firing – i.e. Will can be a verb or a first name.

- person This phase first defines macros for initials, first names, surnames, and endings. This information is then used these to recognise combinations of first names from the previous phase, and surnames from their POS tags or case information. Persons get marked with the annotation 'TempPerson'.

- location Locations are recognised mainly by gazetteer lookup, using not only lists of known places, but also key words such as mountain, lake, river, city etc. Locations are annotated as TempLocation in this phase.

- ambiguities Some ambiguities are resolved immediately in this grammar, while others are left until later phases. For example, a Christian name followed by a possible Location is resolved by default to a person rather than a Location (e.g. 'Ken London'). On the other hand, a Christian name followed by a possible organisation ending is resolved to an Organisation (e.g. 'Alexandra Pottery'), though this is a slightly less sure rule.

- contextual information Although most of the rules involving contextual information are invoked in a much later phase, there are a few which are invoked here, such as "X joined Y" where X is annotated as a Person and Y as an Organization. This is so that both annotations types can be handled at once.

unknown This short grammar finds proper nouns not previously recognised, and gives them an Unknown annotation. This is then used by the namematcher – if an Unknown annotation can be matched with a previously categorised entity, its annotation is changed to that of the matched entity. Any remaining Unknown annotations are useful for debugging purposes, and can also be used as input for additional grammars or processing resources.

name context This grammar looks for Unknown annotations occurring in certain contexts which indicate they might belong to Person. This is a typical example of a grammar that would benefit from learning or automatic context generation, because

useful contexts are (a) hard to find manually and may require large volumes of training data, and (b) often very domain–specific. In this core grammar, we confine the use of contexts to fairly general uses, since this grammar should not be domain–dependent.

date pre This grammar precedes the date phase, because it includes extra context to prevent dates being recognised erroneously in the middle of longer expressions. It mainly treats the case where an expression is already tagged as a Person, but could also be tagged as a date (e.g. 16th Jan).

date This grammar contains the base rules for recognising times and dates. Given the complexity of potential patterns representing such expressions, there are a large number of rules and macros.

Although times and dates can be mutually ambiguous, we try to distinguish between them as early as possible. Dates, times and years are generally tagged separately (as TempDate, TempTime and TempYear respectively) and then recombined to form a final Date annotation in a later phase. This is because dates, times and years can be combined together in many different ways, and also because there can be much ambiguity between the three. For example, 1312 could be a time or a year, while 9-10 could be a span of time or date, or a fixed time or date.

number This grammar covers rules concerning numbers and percentages using the input from the tokenizer and POS-tagger.

price This set of rules is fairly straightforward, using the input from the previous grammar and keywords for currency from the gazetteer lists.

reduced price This grammar contains different grammars that determione if a price is a reduced price and the categories to which this applies.

- child tariff This grammar uses a combination of annotations added by previous grammars, keywords from the gazetteer. The normal and the reduced rate often appear close to each other, so there is a risk of ambiguity. This was however completely reduced by using the word class of the surrounding tokens.

url Rules for email addresses and URL's are in a separate grammar from the other address types, for the simple reason that SpaceTokens need to be identified for these rules to operate, whereas this is not necessary for the other Address types.

event grammar This grammar contains rules to identify various named entities relating events using gazetteer lists.

event person This grammar mainly tries to detect different type of artists, and builds on the grammars name and name context. Some of the classes of named entities these grammars try to discover, tend to be rather stable, that is, a class where most NE are already listed by the gazetteer and there is only a slow growth of new NE. Other classes have a large inflow of new named entities and we can't rely on the gazetteer list to discover them.

- author This grammar fires if a named entity belonging to the class Person is used in combination with certain keywords.

- composer Composers form a rather stable group and are best detected with gazetteer lists.

- musician Musicians are either recognised and annotated by the gazetteer, or are discovered using a series of different patterns. Unless the musician uses an artist name, changes are that it is already annotated as a Person and we can detect it using a series of keywords. Another effective technique is to look for a other musicians in the same sentence. Especially articles on starting and obscure artists often contain a couple of references to well known artists that influenced their music.

- DJ Except for the keyword "DJ" and perhaps some major names the gazetteer is hardly of any use here. We defined the pattern matching rules quite strict to prevent any possible misfires. This grammar only annotates combinations with members of the class Person, or nouns starting with a capital or mixcaps, preceded by the keyword "DJ".

event group This grammar detect different types of groups.

- orchestra We're able to detect most orchestra's by searching for a pattern of ontological classes and keywords. It is for example possible to recognise a great deal of orchestra's looking for a pattern containing a member of the class Location and the keywords "philharmonic" and/or "orchestra".

- music group This grammar annotates music groups either by using gazetteer lists, or discovers new entities using a series of patterns. We assumed that all music group consist of one or more tokens, all starting with a capital letter (we didn't find any contradicting examples in the data set, even though there are probably countless music groups who's name do not follow this specific pattern) in combination with a number of keywords. Another approach used is similar to the one used in musician.jape: Music groups who are still relatively unknown to the the general public are often mentioned together in sentences with other artists as a reference.

event conductor Conductors can best be detected using a combination of certain keywords, the POS-tagger, and the grammar rules to detect persons, or in relation to an orchestra and the grammar rules to detect persons.

event location This grammar detect different types of locations.

- park It is possible to identify almost all parks using the gazetteer, but also without use of gazetteer lists it's straightforward to recognise them checking for combinations of nouns starting with a capital and members of the class Person, both preceding the keyword "park". This grammar also removes previous annotations. This is necessary because parks are often named after a person.

- public square Public squares are recognised in the same fashion as Locations: by gazetteer lookup and using different keywords.

- music hall Most music halls tend to be defined by straight lookup from the gazetteer lists. The remaining music halls can be discovered using lexical information added by the gazetteer lists, the POS-tagger, and previous grammars. A typical example of this would be a sentence containing a music group or musician followed by a verb and the keyword "at". The token or sequence of tokens following this pattern is most likely to be of the type music hall.

- theatre Theatres either get annotated using the gazetteer list or are discovered using patterns involving the keyword "theatre" or other aliases of this keyword.

- museum Most museums are annotated from the gazetteer list or by patterns involving the keyword "museum" in combination added by other grammars (for example, by looking for annotations that are member of the class Person in combinations with the keyword "museum") or information from the POS tagger.

clean This grammar comes last of all, and simply aims to clean up (remove) some of the temporary annotations that may not have been deleted along the way.

After this point all annotations that are considered credible will be transformed into a final named entity annotation.

Simple disambiguation

The result of the previous steps might not have cleared all possible ambiguity concerning certain words due to lack of clues. Some examples:

- The entity “George Washington” should be recognised as a Person (because George is also a first name).
- “U.S. Navy” might be recognised as a “Person” because to the pattern <initials + first letter cap>. This can be disambiguated because the initials also match the label of a location.
- The word “Utrecht” might refer to the mutual exclusive entities “Province” or to “City”. This type of ambiguity can only be resolved by analysing the context of the text.

Named Entity (NE) Co-reference

This component uses pre-generated lists of matching entity notations for the same type.

For example: the annotations for “TU Delft” and “Delft University of Technology” both point to the same entity.

Entity relation extraction

This process determines whether or not a pair of terms in the text has some type(s) of pre-defined relations.

If an entity denoting a concept is already preexisting in the knowledge base and marked as `Trusted`, the annotation system should re-use the reference to this entity. If a concept has a mention in a text, but not yet an entity in the knowledge base, an new entity will be added to the knowledge base. This entity will however not yet be marked as `Trusted`.

Knowledge Base (KB) Enrichment

In the final knowledge acquisition phase, the annotations have to be matched with existing entities in the knowledge base, or, if they do not yet exist, a new entity individual has to be constructed and added to the KB.

Algorithm 4.1 Example of data in KB

```
1 <http://dbpedia.org/resource/Schiphol> <http://proton.semanticweb.
      org/2006/05/protons#mainLabel> "Schiphol" .
2 <http://dbpedia.org/resource/Schiphol> <http://www.w3.org/2000/01/
      rdf-schema#label> "Amsterdam␣Airport" .
3 <http://dbpedia.org/resource/Schiphol> <http://www.w3.org
      /1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/
      Airport> .
```

The newly recognised entity annotations lack instance information and are still not linked to the KB. However these entity annotations could represent entities that are in the recognised part of the KB. The first step is to match the entity annotations by

their class information and string representation against the set of recognised entities. If a matching entity individual is found, the annotation acquires its instance identifier. Otherwise a new entity individual is constructed and added to the KB along with its aliases derived from the list of matching entities (if such). At this point all generated named entity annotations are linked to the ontology (via their type information) and to the KB (via their specific instance). The relation annotations generated by the template relation extraction grammars, are used to generate the according entity relations in the KB (e.g.: person's positions; spatial positioning information for organizations, etc.). This finalises the annotation process, having as a result named entity annotations linked to their semantic descriptions in the KB.

4.2 Data layer

This layer is the home of the ontology and provides a simplified access to both the semantic repository and the Lucene Index.

Ontology

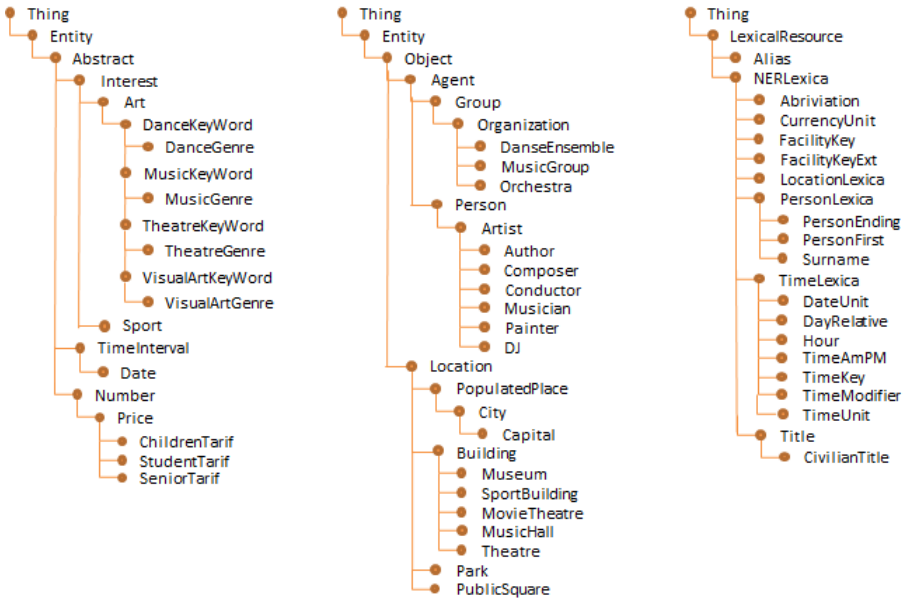

Figure 4.3: Ontology of the Event Annotation System

Section 2.1.2 argued that ontologies used in Natural Language Processing should model, next to lexical knowledge, domain knowledge that can easily be re-used across different domains by different agents. However, before drafting up an ontology, we first analysed over a six hundred articles. This led to the observation that almost all descriptions of an event contain the following bits of information: something that describes an interest, a group or an individual that performs, a location, and the time and

date that the event takes place. The proposed ontology is based on PROTON (PROTo ONtology) [16] which is a basic subsumption hierarchy and provides broad coverage of many of the upper-level concepts necessary for semantic annotation, indexing, and retrieval. This ontology was extended with the necessary classes.

- Abstract: Intangible objects that describe some aspect of an event. This is the parent class of concepts such as Interest (e.g. somebody can be interested in "Dance Music" or "Opera" which are individuals respectively belonging to the class DanceGenre and MusicGenre), Time and Date, and Price.
- Agent: An Agent is something, which can show (carry out) an independent action, whether consciously or not. This can be an organization (DanceEnsmble, MusicGroup, and Ochestra) or a single artist (Author, Composer, Conductor, Painter, etc).
- Location: This is the parent class of all Things that have a (fixed) geographic location on the earth. Relevant sub-classes are City (and Capital), Building (Museum, SportBuilding, MovieTheatre, etc), Park, and PublicSquare.
- LexicalResource: A lexical resource of any sort, usually part of a natural language or a specialised vocabulary used to determine the context.

DBpedia Connector

Creating gazetteer lists and keeping them up-to-date can be a laborious task. Therefore we developed a component that synchronises the named entities in the semantic repository with relevant data from DBpedia[2]. This is a project aiming to extract structured content from the information available on Wikipedia[3], and making this data accessible using the SPARQL-query language. Because of this, it becomes possible to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data.

We've also extracted data from DBpedia that is not directly useful for NER, but might come in handy to build future applications on top of the proposed platform. For example, were available we've added coordination of the location based entities to the semantic repository. This data might be useful in a later stage to plot results on a map.

Lucene Index Store

Documents and their annotations are stored in a Lucene index store. An index stores statistics about terms in order to make term-based search more efficient. Lucene's index falls into the family of indexes known as an inverted index. The articles added to the Lucene Index store are indexed on the named entities that appear in the text.

[2] http://dbpedia.org/
[3] http://en.wikipedia.org/wiki/Main_Page

SAIL API

BigOWLIM implements the Sesame SAIL interface so that it can be integrated with the rest of the Sesame framework.

Semantic Repository

BigOWLIM is used as a Semantic Repository. According to Ontotext it is the most scalable semantic repository in the world, and delivers an outstanding multi-user query performance.

4.3 Semantic search system

Even though the implementation of the semantic search system is outside the scope of the Master Thesis, we have developed a working version of a (very) basic semantic search system, which is able to retrieve annotated documents based on named entities contained in these documents. This system is developed on top op the Apache Tomcat framework[4], making use of JavaServer Pages (JSP).

4.3.1 Search mechanism layer

This layer groups the different mechanisms on which the semantic search services are built.

Query builder

The search process starts with the user constructing a query that reflects his or her information needs. Having chosen at least one facet to be represented as node in the graph, it can be used to build semantically unique search queries. By selecting one of the objects in a facet, the result set can be filtered to only objects that are directly or indirectly connected to the selected one.

Query dispatcher

The function of the Dispatcher is to route the query request to the repository.

Result parser

The result parser accepts XML data and transfers this to a native Java data type.

[4] http://tomcat.apache.org/

4.3.2 Semantic search services

The Search Services Layer groups the different functionalities a semantic search engine offers to the user.

Facet ranking

There are three aspects to consider when it comes to the presentation of the results [29]: selecting what data to present, organising the results, and ordering the results. Facted browsing allows the user to constrain the set of results within a particular facet. Typically, facets are directly mapped to properties in RDF, or, alternatively, the mapping is made by projection rules. In case of a large data set, the number of facets will typically also be very large and a mechanism to automatically rank the facets is required. A facet typically represents one important characteristics, therefore, in our casus, predicates are probably most suitable for this task.

Action listener

Reacts to the user‘s request by selecting the appropriate action.

4.3.3 Presentation layer

This layer provides for the interaction between the user and the system.

Presentation Logic / Front-End

The front-end contains the basic components for the user to create a search query and to display the results of such a query. The basic interface components for the user to construct such query are: text entry boxes, value selection lists, and tree structures to display and select facets. Given the geographical nature of the data, results are best displayed on a map. Then, using facets, the user can narrow the number of information-items, in which he is actually interested in, further down.

4.4 Conclusion

We designed an architecture for a semantic annotation system and semantic search engine, together consisting of six distinctive layers. Each layer unambiguously shields off implementation details, and has a defined set of functionalities, required input, and an expected outcome. These layers are: semantic data acquisition, knowledge acquisition, data layer, semantic search mechanism, semantic search services, and results presentation layer, We’ve implemented a working solution of the three layers, capable of crawling, annotating and storing event-related articles.

The crawler contains a set of URIs as input and determines the content type of each URI. Based on that a suitable InputParser is selected. The InputParser contains the

most logic of all the components in the data acquisition layer. It determines for example if a document or feed-item gets annotated, if the links contained in a document should also be retrieved, and if the annotated document should be discarded or stored in the data layer.

The knowledge acquisition layer is implemented using the GATE platform. First a tokenizer splits the text in very simple tokens. The second component, the gazetteer, loads the aliases of all the entity descriptions that are available to it via the semantic repository, and annotates those entities in the text for which it can find a matching alias. Next, a sentence splitter groups all tokens into sentences using regular expressions and based on that input, a part-of-speech (POS) tagger determines per token the correct word class.

Documents and their annotations are stored in a Lucene Index, whereby documents are indexed based on the annotations they contain which makes for easy retrieval. Because creating gazetteer lists and keeping them up-to-date can be a laborious task, the data layer also contains component that synchronises the named entities in the semantic repository with relevant data from DBpedia – an addition that significantly increases the knowledge of system with new semantic descriptions of named entities.

Chapter 5

Evaluation

Three different annotation strategies were tested using the annotation pipeline described in the previous chapter (see section 4.1.2). This chapter will first introduce these annotation strategies and next evaluate their quality in accordance accordance to the research question asked in section 1.2. To do this, we will present the metrics to measure the overall performance of the annotation system, and, using these metrics, we will answer the following sub-reseach questions:

1. How well does the system perform at annotating relevant concepts in a text?
2. How does our solution perform compared to current annotation systems?
3. Does the use of pattern matching grammars also yield more accurate annotations?

The first two questions discuss the performance of the produced system, while the third question is more focused on the usability of pattern matching grammars in general.

The rest of this chapter is structured as follows: First, section 5.1 will discuss the evaluation methodology. Next, section 5.2 will describe the used data set. Section 5.3 discusses the different annotation startegies. Section 5.4 will go into the results, and section 5.6 will present our evaluations based on these results.

5.1 Methodology and metrics

As discussed in Chapter 3, semantic annotation is about assigning named entities in texts with links to their semantic description. This annotated meta-data should provide a class and a reference to an entity in a semantic database. To measure the correctness of our machine produced annotations, we should evaluate each of these three aspects of an annotation and compare them to a ground truth – a human produced set of annotations, which is considered to be correct. The annotations produced over a corpus by the automated system are called the *response set*, while the annotations that make up the ground truth are known as the *key set*. The following definitions express some

of the different possible relationships between annotations from the key set and the response set, adapted for use with semantic annotations from [26]:

- Coextensive: Two annotations are coextensive if they hit the same span of text in a document. That is, both their start and end offsets are equal.
- Overlaps: Two annotations overlap if they share a common span of text.
- Compatible: Two annotations are compatible if they are coextensive and they both have a reference to the same entity in the knowledge base and share the same semantic class.
- Partially Compatible: Two annotations are partially compatible if they overlap in span, if the class of response annotation is equal to the class of the key annotation, or a super type of the class of the key annotation.

Building upon these definitions, we can now draw a comparison between the annotations in the key set and the response set:

- Correct: An annotation from the response set is considered correct, if it is compatible with an annotation in the key set.
- Partially correct: An annotation in the response set is partially correct if it is partially compatible with an annotation from the key set.
- Missing: An annotation is missing in the response set if it there is no coextensive or overlapping response annotation for an annotation in the key set.
- Spurious[1]: A response annotation is spurious if it is either not coextensive or overlapping with an annotation in the key set, or, the class of the response annotation is not equal nor a super class of the key annotation.

All annotations are considered individually. If a certain concept has several mentions in a text, but some of them are not correctly annotated or are missing, these mentions will be respectively marked as Partially Correct, Missing or Spurious – this despite the fact that the concept might be correctly recognised elsewhere in the same text.

[1]Spurious is a synonym for false or fake, bastard. However, in the context of annotations, another, more common, term for spurious would be "false positive". It is an annotation in the reference set that has no equivalent in the key set.

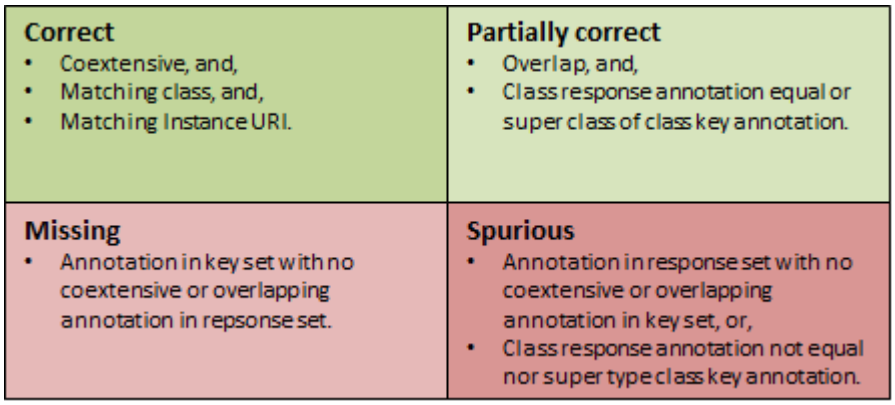

Figure 5.1: Compared to the annotations in the key set, a response annotation falls into one of these categories.

Using these definitions, we can now perform a qualitative analysis and calculate the following metrics:

Precision

Precision measures the number of correctly identified items as a percentage of the number of items identified. In other words, it measures how many of the items that the system identified where correct, regardless of whether it also failed to retrieve other correct items.

$$PrecisionStrict = \frac{Correct}{Correct + Partial + Spurious}$$

$$PrecisionLenient = \frac{Correct + Partial}{Correct + Partial + Spurious}$$

Recall

Recall is the number of correctly identified items in the response set as a percentage of the total number of correct items in the key set. It is a measurement that indicates how many items that should have been identified were actually identified. This regardless of how many spurious identifications were made.

$$RecallStrict = \frac{Correct}{Correct + Missing}$$

$$RecallLenient = \frac{Correct + Partial}{Correct + Partial + Missing}$$

False Positives

False Positives measures the number of incorrectly identified items as a percentage of total number of tokens in the text, here indicated by c.

$$FalsePositives = \frac{Spurious}{c}$$

F-measure

F-measure is the weighted average between Precision and Recall. This combined score makes it possible to order annotation results.

$$F\text{–}measureStrict = \frac{2 * PrecisionStrict * RecallStrict}{PrecisionStrict + RecallStrict}$$

$$F\text{–}measureLenient = \frac{2 * PrecisionLenient * RecallLenient}{PrecisionLenient + RecallLenient}$$

5.2 Dataset

The data set is composed of articles published on the website iamsterdam[2]. This website contains a great number of English articles on events in and around Amsterdam. Since these articles are published by a variety of contributors and independent of the site's redaction, they can greatly differ in ways to describe certain concepts and are subject to changing writing styles and information density.

From every article on the site we've copied its title, url and the text describing the event. This resulted in a data set of close to 650 articles. By manually analysing each of these articles we identified the most important concepts mentioned. An article describing an event often contains a combination of mentions of the following concepts:

- Location: An event takes place at one or more locations. A location can be a MusicHall, Theatre, MovieTheatre, Stadium, Park, Square, etc.
- Agent: An agent is a person or a group. Often an agent is the performing artist or the artist on display.
- Date: An event takes place over the course of one or more dates.
- Price: The price of the event. It might be so that different rates apply for children or other special interest groups eligible for a discount.
- Keywords: Specific keywords that relate to a type of event or location (e.g. theatre, exhibition, concert, park, etc).

[2] http://www.iamsterdam.com/nl/visiting

Twenty articles where randomly selected from this data set for evaluation purposes. These articles were not used during the development phase, in order to make sure not to influence the evaluation results. Next, these articles were manually annotated in order to create a key set of annotations against which we could compare the results of the three different annotation systems. An example of an annotated text for the key set can be seen in Figure 5.2. An overview of the entire evaluation-dataset and its most important characteristics are described in Table 5.1.

Seoul Philharmonic Orchestra (Unannotated)
Famed maestro Myung-Whun Chung conducts his Seoul Orchestra in a performance of works by Ravel (’La valse’), Tchaikovsky (’Pathetique’ Symphony) and Debussy (’La mer’). Company: Seoul Philharmonic Orchestra at the Concertgebouw on Fri 19 Aug 2011 20:15. Prices range from 24, - to 34,-; Children: 24,-

Seoul Philharmonic Orchestra (Annotated)
Famed maestro [Myung-Whun Chung : class=”../Conductor”, inst=”../Myung-whun_Chung”] conducts his [Seoul Orchestra : class=“../ Orchestra”, inst=”../Seoul_Philharmonic_Orchestra”] in a performance of works by [Ravel : class=../”Composer”, inst=”../Maurice_Ravel”] (’La valse’), [Tchaikovsky: class=../”Composer”, inst=”../Pyotr_Ilyich_Tchaikovsky”] (’Pathetique’ Symphony) and [Debussy : class=”../Composer”, inst=”../Claude_Debussy”] (’La mer’). Company: [Seoul Philharmonic Orchestra : class=“../Orchestra”, inst=”../Seoul_Philharmonic_Orchestra”] at the [Concertgebouw : class=”../MusicHall”, inst=”../Concertgebouw”] on Fri [19 Aug 2011 : class=../Date””, inst=”../Date_19_Aug_2011”] [20:15 : class=”../TimeInterval”, inst=”../TimeInterval_20%3A15”]. Prices range from [24 : class=”../Price”, inst=”../Price_24”],- to [34 : class=”../Price”, inst=”../Price_34],-; Children: [24 : class=”../PriceChildren”, inst=”../PriceChildren_24],-

Figure 5.2: Example Key Set

	Title Text	Word Count	# Annotations Key Set
1.	Sunday Market	204	6
2.	Biosfeer	172	3
3.	Seoul Philharmonic Orchestra	109	15
4.	Adam Fuss	269	9
5.	Cage the Elephant	72	6
6.	Klinch	234	11
7.	Kitty, Daisy & Lewis	194	12
9.	Metropolitan Opera	189	7
9.	Open Podium	140	23
10.	The war on drugs	129	8
11.	The Karnatic Lab	332	6
12.	The drums	144	9
13.	The Magic Flute	300	8
14.	Open Monumentendag	518	79
15.	PICNIC	381	5
16.	Amsterdam Roots Festival	499	11
17.	Silent Disco	386	37
18.	Van Dik Hout	113	12
19.	Sinead O'Connor	193	12
20.	Junip	73	2

Table 5.1: The dataset used for the evaluation and its most important characteristics. It can be seen that the different articles vary greatly in length, subject and number of annotations in the key set.

5.3 Annotation strategies

This section introduces the three different annotation strategies. The three different strategies that were used to annotate this corpus are:

- General Concept Strategy: This strategy can only detect general concepts such as People, Organizations, Locations, and, Time. It is composed of readily available pattern matching grammar rules and has a knowledge base of 80.000 instances on the before mentioned concepts. This strategy represents the current annotation strategy and serves as the baseline.

- Extended KB Strategy: This strategy is built upon the General Concept Strategy by extending the knowledge base with 30.000 domain specific instances derived from DBpedia[3]. Also we have removed the most erroneous grammar rules. This strategy has the advantage that it can be implemented relatively fast. If a knowledge base is present and the right ontology has been determined.

[3]http://dbpedia.org

- Domain Grammar Strategy: It has access to the same instances in the knowledge base as the Extended KB strategy, but has custom built pattern matching grammar rules to detect domain specific concepts that are not included in the knowledge base. Challenge here is to create the right set of grammar rules, meaning annotating as many meaningful concepts as possible while at the same time minimising the number of spurious annotations.

5.4 Results overview

We evaluated the performance of the three annotation strategies against a ground truth. All the three different annotation strategies make use of the same underlying ontology. It can be possible that a strategy is not always able to assign the most the most accurate ontological class to a concept, but is however able to identify it as a member of a parent class. Therefore it was decided to also include a lenient measurement in the evaluation of the results as described in Section 5.1.

Table 5.2 shows the results for the three different annotation strategies.

Strategy	Precision		Recall		False Pos.	F– measure	
	Strict	Lenient	Strict	Lenient		Strict	Lenient
General Concept	0.527	0.74	0.509	0.589	0.057	0.453	0.628
Extended KB	0.711	0.876	0.768	0.806	0.011	0.711	0.797
Domain Grammar	0.818	0.887	0.914	0.915	0.007	0.850	0.895

Table 5.2: Evaluation metrics per strategy

The difference between the strict and lenient results for the precision of the General Concept Strategy demonstrates that this strategy performs fairly well at recognising general concepts, but lacks the knowledge to produce more precise results. For example, if we take the sentence “Rock legend Bruce Springsteen will promote his new album with a series of concerts.”, the named entity “Bruce Springsteen” will be recognised as a Person, but not as a Musician – which would be considered more correct since we’re annotating articles on events. Because the grammars in General Concept Strategy are originally optimised for articles from the business pages of the Wall Street Journal, we see that this strategy also produces a relatively high number of false positives.

The most erroneous grammar rules the where removed for the Extended KB strategy and the knowledge base was extended with 30.000 domain specific instances. It should be obvious from Table 5.2 that these relatively simple adjustments already resulted in a sharp decline in false positives. Another observation is that the addition of domain specific instances to the knowledge base yields a strong improvement in results for both precision and recall. If we compare the f–measure of the Extended KB strategy with that of the baseline an improvement of nearly 60% is visible.

The third strategy combines domain specific instances in the knowledge base with the flexibility of grammar rules. It can put forward strong results for both precision and

recall. This resulted in an 80% better performance if we compare its f–measure to the baseline. This strategy produces produces 8 times less false positives than the baseline and (almost) half of those produced by the Extended KB strategy.

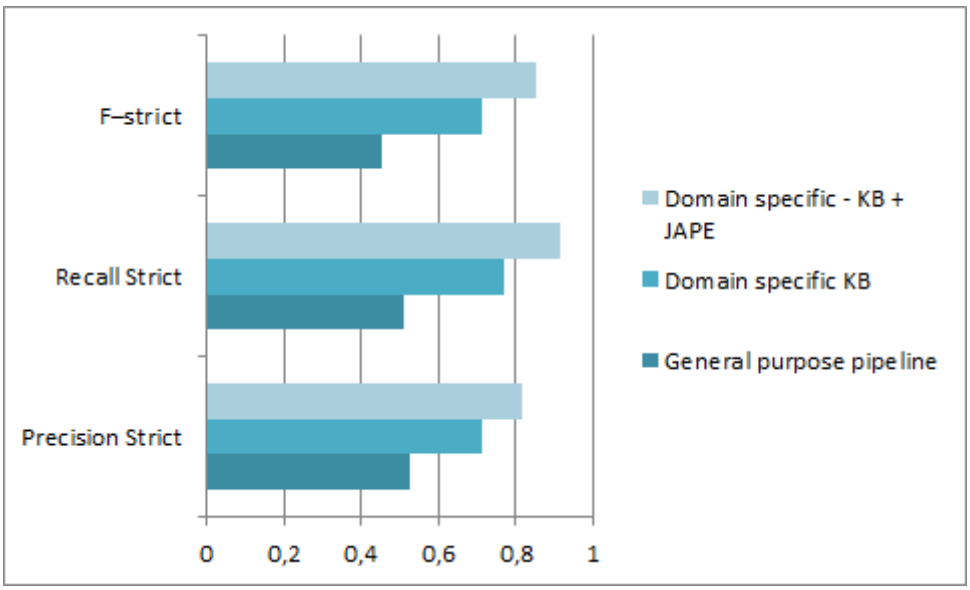

Figure 5.3: Results Strict Evaluation

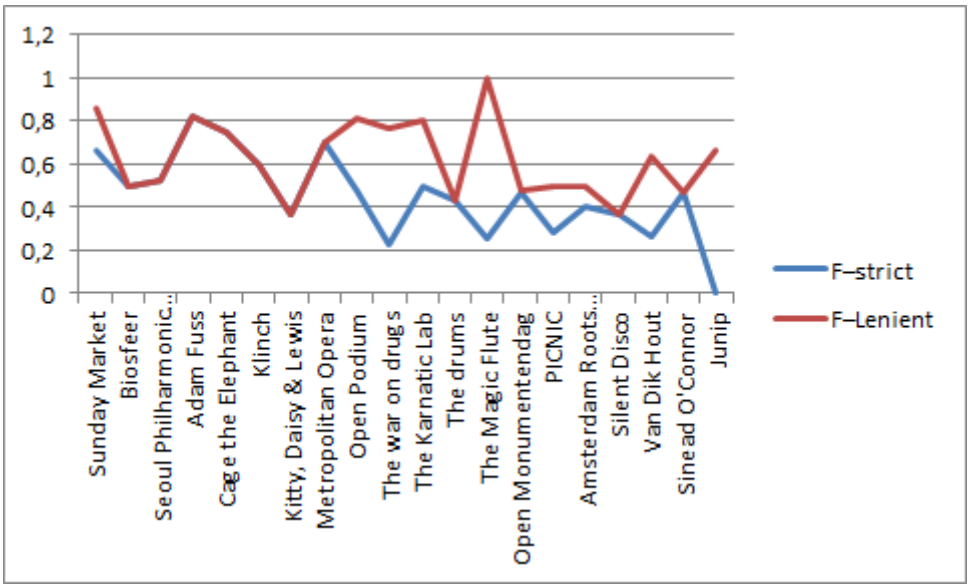

Figure 5.4: F-measure General Purpose Pipeline

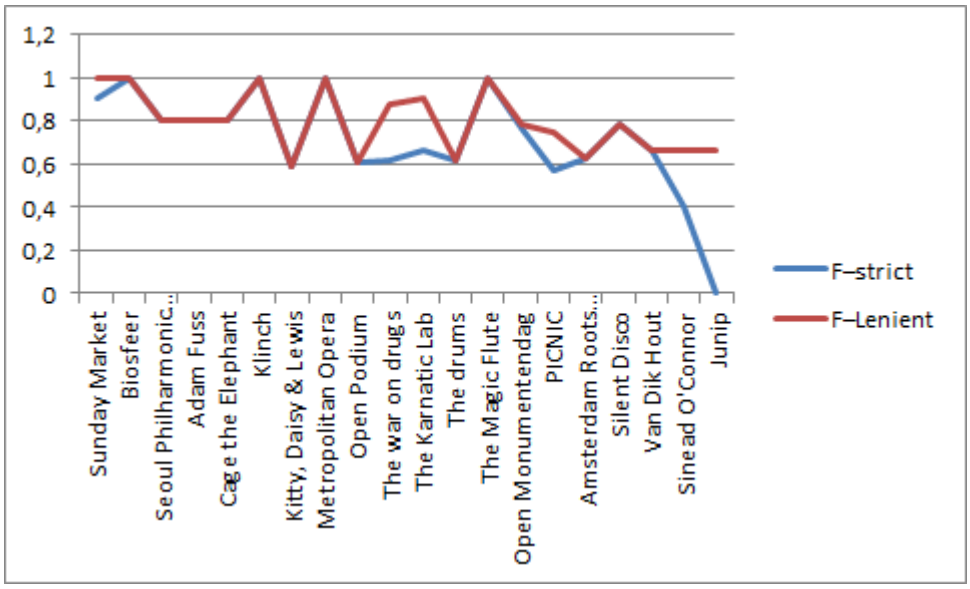

Figure 5.5: F-measure KB

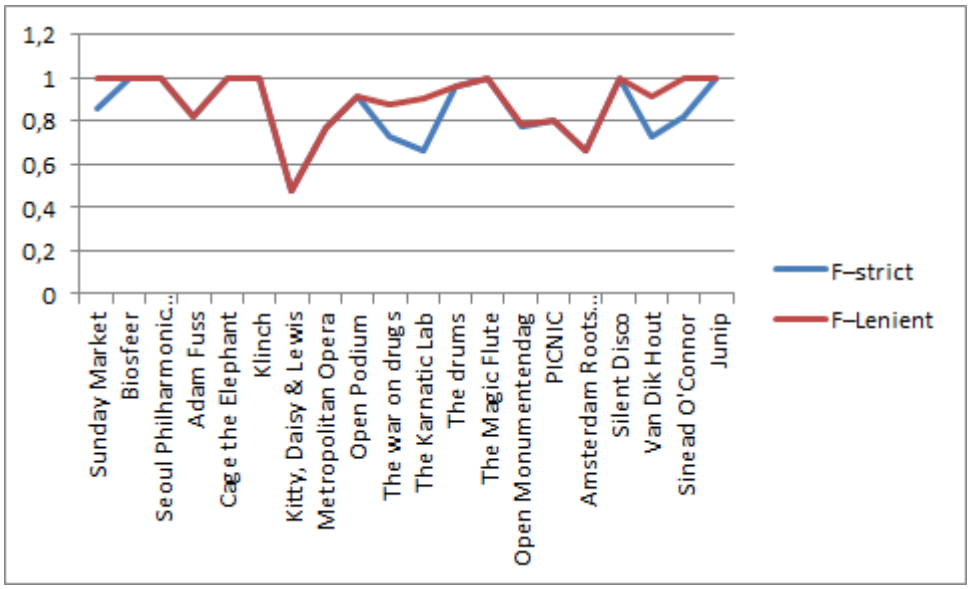

Figure 5.6: F-measure KB and Grammar rules

5.5 Analysis of results

Following the discussion of the evaluation results in the previous section, we will now focus more closely on the different factors that influenced these numbers.

Ambiguous terms

It can occur that part of a text gets wrongfully linked to an instance in the knowledge base because it matches its literal value without actually being semantically related to it.

Example1

Suppose the knowledge base contains the following two tupples:

Algorithm 5.1 Semantic description of the band "Free"

```
1 <http://dbpedia.org/resource/Free_(band)> <http://www.w3.org/1999/02/22-rdf-
      syntax-ns#type> <http://EventAnnotations.com/Ontology/proton-EA#MusicGroup
      > .
2 <http://dbpedia.org/resource/Free_(band)> <http://proton.semanticweb.org
      /2006/05/protons#mainLabel> "Free" .
```

This will lead the annotation system to conclude that every mention of the word `Free` is related to this particular music group, which results in the following annotations:

> The festival terrain on [Beursplein] is open from [11:00] to [20:00] on both days. Tickets: [free : class="../MusicGroup", inst="../Free_(band)"] entrance.

or

> The [Westergasfabriek] in [Amsterdam] presents the Sunday Market - a funky, [free : class="../MusicGroup", inst="../Free_(band)"] event for all market lovers!

Since the same term is used here for different concepts, it leads to spurious annotation and has a negative effect on the precision of the results,

Pattern mismatch

Using grammar rules it is possible to describe certain patterns, and than hopefully this pattern indicates a mention of a mention of a concept in a text. It can however occur that that there is a mismatch and a grammar rule wrongfully fires. Consider the following two examples:

Example2

> Combined with [vocals] and a layer of [electronica] and the [music] of [New York] band [The Drums] becomes reminiscent of the [Animal Collective], while hints of [Joy Division : class"../Person", inst="../Joy_Division"] and [The Cure] also creep through in some [songs].

In this example the gazetteer component first annotated the token "Joy" as belonging to the class PersonFirstFemale. This in turn fired a grammar rule that stated that all combinations of [PersonFirstFemale] + [Token.orth = upperInitial] are members of the class Person. This is a spurious annotation and has a negative effect on the precision of the results.

Example3

> This dance performance by [De Movers] at the [Martin Luther King {Park : class= "...", inst="../Park"} : class="../Person", inst="../Martin_Luther_King_Park"], tells the story of three friends going on an adventure in search of happiness.

The two tokens Martin and Luther both get annotated as members of the class PersonFirst. Since these tokens are followed by two tokens starting with upper initials the entire sequence, Martin Luther King Park, got wrongfully annotated as a member of the class Person. This is a spurious annotation and has a negative effect on the precision of the results.

Parks, but also other types of locations, are often a combination of name and a key word that is specific for the class it belongs to. A list of terms denoting locations was added to the knowledge base. This new information can then be combined with a grammar rule that fires for all combinations of a Person + KeyWord and annotate them accordingly.

Missing conceptual information

If conceptual information is missing, it becomes infeasible to detect and annotate the concept. A good pattern matching rule should not be tailored just to detect one mention and should also cause as few misfires as possible.

Example4

> The American band [War on Drugs] are on the Secretly Canadian label, which also put out [Anthony and the Johnsons] and [Jens Lekman].

The system was able to recognise "War on Drugs" as a MusicGroup. using the contextual It was however not able to recognise "Anthony and the Johnsons" as a MusicGroup and "Jens Lekman" as a Musician because the right conceptual information is missing to do so. All three mentions where annotated as a Person, which is partially correct for "Jens Lekman", but negatively influencing the results for *PrecisionStrict*. The annotations "Anthony" and "Johnsons" are however both spurious, and the annotation [Anthony and the Johnsons : class="MusicGroup"] is missing.

Example5

> Dublin-born [Sinead O'Conner : class="../Person", inst="../Sinead_O%27conner"] made a name for herself in the early 90s.

The above text does not contain enough contextual information to annotate "Sinead O'Conner" as a Musician. It is however possible to classify her as a member of the (parent-) class Person, an annotation which is only partially correct, and therefore negatively influences *PrecisionStrict*.

With more conceptual information in the text, or using prior knowledge the annotation system

> Dublin-born singer [Sinead O'Conner : class="../Musician", inst="../Sinead_O%27conner"] made a name for herself in the early 90s.

If she would have been pre-defined as a musician in the knowledge base.

Algorithm 5.2 Semantic description of "Sinead O' Conner"

```
1 <http://dbpedia.org/resource/Sinéad_O'Connor> <http://www.w3.org/1999/02/22-
      rdf-syntax-ns#type> <http://EventAnnotations.com/Ontology/proton-EA#
      Musician> .
2 <http://dbpedia.org/resource/Sinéad_O'Connor> <http://proton.semanticweb.org
      /2006/05/protons#mainLabel> "Sinead O'Conner" .
```

Co-referencing

Mentions of the same concept should all reference to the same entity in the knowledge base, otherwise the annotation will only be considered partially correct. When a concept has a mention in a text, but not yet an entity in the knowledge base, an new entity will be added to the knowledge base. This entity will however not yet be marked as Trusted.

Example6

> Die Zauberflöte is a Dutch translation of [Mozart : class="../Composer", inst="../Wolfgang_Mozart"]'s most famous opera. [Mozart : class="../Composer", inst="../Wolfgang_Mozart"] did not write The Magic Flute (Die Zauberflöte) for a big opera theatre as we are now accustomed to, but for a small theatre in a suburb of Vienna.

"Mozart" is already defined as entity in the KB and belongs to the class Composer. The gazetteer will reuse the reference to this entity for both mentions.

Example7

The knowledge base contains an entity for the concept "New York City" with the following labels:

Algorithm 5.3 Different aliases for the named entitiy representing "New York"

```
<http://dbpedia.org/resource/New_York_City> <http://proton.semanticweb.org
    /2006/05/protons#mainLabel> "New York" .
<http://dbpedia.org/resource/New_York_City> <http://www.w3.org/2000/01/rdf-
    schema#label> "Big Apple" .
<http://dbpedia.org/resource/New_York_City> <http://www.w3.org/2000/01/rdf-
    schema#label> "The City That Never Sleeps" .
<http://dbpedia.org/resource/New_York_City> <http://www.w3.org/2000/01/rdf-
    schema#label> "Gotham" .
```

These labels can be used by the gazetteer to correctly reference the mention "`New York`" and its synonym "Big Apple".

> Pathé Opera sees a series of Metropolitan Opera performances from [New York : class="../City", inst="../New_York_City"] streamed into the Amsterdam's Pathé Tuschinski cinema. [..] And thanks to the latest audio-visual technology and use of close ups, you'll probably get an even better view of the show than the audience in the [Big Apple : class="../City", inst="../New_York_City"]!

Example8

The above two examples required prior knowledge that was already contained in KB. The example below makes use of pattern matching rule

Algorithm 5.4 JAPE pattern matching rule for discovering an orchestra

```
(
    (
        ({Token.kind == word, Token.category == NNP, Token.orth ==
            upperInitial})+
        ({Token.string == "Orchestra"})
    )
    |
    (
        ({Token.kind == word, Token.category == NNP, Token.orth ==
            upperInitial})+
        ({Token.string == "Orchestra"})
        ({Token.kind == word, Token.category == IN})
        ({Token.kind == word, Token.category == DT})
        ({Token.kind == word, Token.category == NNP, Token.orth ==
            upperInitial})
    )
):orchestra_
```

This will result in two annotations for the concept "Seoul Philharmonic Orchestra" in the text below. The system is however unable to co-reference the mention "`Seoul Orchestra`" with "`Seoul Philiharmonic Orchestra`". The annotation "`Seoul Orchestra`" can therefore only be considered *PartiallyCorrect*.

> Famed maestro Myung-Whun Chung conducts his [Seoul Orchestra : class="../Orchestra", inst="../Seoul_Orchestra"] [..]. Company: [Seoul Philharmonic Orchestra :: class="../ Orchestra", inst="../Seoul_Philharmonic_Orchestra"] at the Concertgebouw [..].

Solving this co-referencing problem is however outside the scope of this MSc thesis.

Unknown concept

If a concept is not in the knowledge base and there is no grammar rule either to detect its mention in some text, the missing annotation will have a negative effect on the recall.

> Graffiti6 is a refreshing project that unites the unlikely pairing of producer TommyD and singer-songwriter Jamie Scott.

Producer is an unknown concept to the system and Tommy D will at best be detected and annotated as an instance belonging to the class Person.

5.6 Synopis

At the beginning of this chapter a couple of questions were formulated. The first two questions mainly relate to the quality of the proposed annotation system, while the third question is more focused on the usability of pattern matching grammars in general. Each of these questions will be discussed below.

1. **How well does the system perform at annotating relevant concepts in a text?**

   The annotation system can put forward strong results for both precision and recall, and only produces a few false positives. For example, the best strategy performs with a precision of 0.818 and a recall of 0.914. The system is able to discover and annotate almost all of the entities which it is expected to annotate and achieves this goal almost without adding any spurious annotations in the process.

2. **How does our solution perform compared to current annotation systems?**

   When comparing the F-measure of the proposed annotation system against the baseline, table 5.2 shows that the first outperforms the latter with about 80%. If we compare the results between the baseline strategy and Domain Grammar strategy, we see that there is a difference of 0.29 points for the strict evaluation, but only 0.14 points difference for the lenient evaluation. This indicates that the baseline performs fairly well at annotating those named entities it does find with the right parent class, but lacks sufficient knowledge to produce more precise results. The contrast is much larger when we compare the two strategies on recall,

with only a negligible difference between a strict and lenient evaluation. The general concept strategy fails to annotate a fair amount of named entities due to lack of knowledge about those entities. Furthermore, the baseline strategy also produces a large amount of false positives, most of which consisted of irrelevant annotations.

The above numbers demonstrate that the proposed system clearly outperforms the current one when it comes to annotating event-related articles. Even though the grammars from the General Concept strategy, which are originally optimised for annotating business articles, form a somewhat suitable basis for annotating events, it should be noticed that these rules produce a large number of false positives and missing annotations in a different context. The most decisive factor however is the absence of a domain specific grammar and instances.

3. **Does the use of pattern matching grammars also yields more accurate annotations?**

   Pattern matching grammars add flexibility to the system. Whereas gazetteer lists can only annotate named entities that have a matching instance contained in the knowledge base, pattern matching grammars can put forward the following advantages: They can discover new mentions of concepts and at this knowledge to the system. To do this, they combine keywords and information from the POS-tagger into a set of regular expressions. This effect can also be seen in the evaluation results. For recall there is a difference of 0.14 points (strict) and 0.109 (lenient) between the Extended KB strategy and the domain grammar strategy; Pattern matching grammars can add more precision to the annotation results. The Extended KB strategy has a (strict) precision of 0.711 whereas the Domain Grammar strategy has 0.818, which is a difference 0.107 points. The precision can be increased either by the use of a keyword which reveals the annotation belongs to a more detailed subclass (e.g.: "[Alison Goldfrapp : class="Person"] and her [band : class="MusicKeyword"]". With this extra information Alision Goldfrapp can now be annotated as a Musician) or because it appears in a list of elements belonging to the same class (e.g.: "For their music they draw inspiration from [U2 : class="MusicGroup"], [Bloc Party : class="Unknown"], and [Moloko : class="MusicGroup"]"'. Bloc Party has already been labelled as potentially interesting by a previous grammar rule and can now be identified as a MusicGroup); JAPE Pattern matching grammars remove annotations. This sort of functionality can be used to add temporary annotations (see "Bloc Party" labelled as Unknown in previous example), or, if the context is ambiguous, add multiple annotations to the same span of text and decide at a later stage. The use of domain specific grammar rules resulted in almost half less false positives if we compare the Extended KB strategy to the Domain Grammar strategy.

Part III

Conclusions and future work

Chapter 6

Conclusions and Future Work

This chapter will first summarize the answers to the different research questions and the different contributions made in the course of this Master's Thesis. Next, we conclude this chapter with an overview of future work.

6.1 Conclusions

Conventional search techniques are developed on the basis of words computation models and link analysis. They do not understand the meaning of words and are thus ill equipped to handle complex queries. There are however two approaches that would enable computers understand human produced texts: an intelligent computer system which can interpret human languages, or machine-readable information can be added to the data on the Web. The latter approach is applied by the different technologies that make up the Semantic Web and extends the scope of traditional information retrieval paradigms from mere document retrieval to entity and knowledge retrieval. It improves the conventional IR methods by looking at a different perspective, namely, the meaning of words, which can be formalised and represented in machine processable format using ontology languages such as RDF and OWL. By including a logical representation of resources, a semantic search system is able to retrieve meaningful results by drawing inference on the query and knowledge base.

Because the World Wide Web contains a vast and ever growing collection of documents, it is impossible to manually annotate every single one of them. An automated annotation method provides the scalability needed to annotate existing documents, reduces the burden of annotating new documents, or allows for the use for the use of multiple ontologies which can be beneficial to support the needs of different users.

During the course of this thesis we explored the necessary steps to realise a semantic search engine for location-based data on events, and proposed a high-end architecture to this means. This architecture consists of six distinctive layers, whereby each layer represents an unambiguous set of defined functionalities with a required input and an expected outcome. These layers are: semantic data acquisition, knowledge acquisition, data layer, semantic search mechanism, semantic search services, and result

presentation layer. The combination of these layers form two logical systems: the first pair layers form a semantic annotation system, while the combination of the last three layers form a semantic search engine. A data layer for storing and retrieving documents and their respective annotations, plus a semantic repository containing a large collection of instances, is shared between these two logical systems.

We have a working implementation of the annotation system – that is, we implemented a functional crawler, annotation pipeline capable of discovering event-related named entities in documents, and storage system. For designing the annotation pipeline, we analysed over a hundreds of event-related articles and from this we identified a number of key concepts, and placed them in OWL ontology.

We implemented a custom build annotation pipeline, making use of the well-known GATE platform for discovering named entities, and assigning them with their semantic class and a unique instance URI.

To detect named entities in texts that are already known, we make use of gazetteer lists. Because making and maintaining these lists can be quite a laborious task, we've implemented a process which synchronises the semantic repository with DBpedia. This solution also allows us to expand the knowledge in the system with external knowledge on newly discovered named entities, for example, adding a longitude and latitude to discovered NE of locations (given that it is provided for in DBpedia). Unknown named entities are discovered using a combination of keywords and word classes, respectively added by the gazetteer and the part-of-speech tagger. By combining these elements in manually crafted JAPE-grammar rules, the system is capable of discovering named entities which are not already part of the system's knowledge.

Documents and their annotations are stored in a Lucene Index, whereby documents are indexed based on the annotations they contain which makes for easy retrieval.

We evaluated how well the proposed annotation system performed at annotating NE in texts (on a different corpus than the one corpus used in the development face) and compared these results to a ground truth. This gave respectively the following strict and lenient results: 0.818 / 0.887 for precision, 0.914 / 0.915 for recall, and 0.850 / 0.895 for the F-measure. These figures demonstrate that the annotation system can put forward strong results for both precision and recall, and only produces a few false positives. In other words, the system is able to discover and annotate almost all of the entities which it is expected to annotate and does this without adding to many spurious annotations in the process. Whenever mistakes were made, either because of a spurious or a missing annotation, this was mainly because the system was not able to correctly interpret the available and often ambiguous conceptual information, or not enough conceptual information was present.

If we compare the results of the current system to the results of the baseline by means of the F-measure, it should be obvious that the proposed system outperforms the current system with roughly 80% better annotation results. The baseline system performs fairly well at annotating named entities with its right (parent) class, but often lacks sufficient knowledge to produce more precise results. The contrast is much larger when we evaluate the two systems on recall. Named entities are recognised either because they already exist in the knowledge base, or the system discovers them using a combining word classes and keywords into a set of grammar rules. There are a couple of

reasons to why the baseline system under performs compared to the proposed system: its grammars and knowledge base are optimised for the domain of business articles and lack the precision to annotate concepts that were identified as imported for the domain of event-related articles. Furthermore, the baseline strategy also produces a large amount of false positives, which consisted for a large part of irrelevant annotations.

The proposed solution makes use of pattern matching grammars. Pattern matching grammars make the annotation process more flexible because they can add and remove annotations, functionality which can be used to resolve ambiguity, and they increase precision of the results.

In conclusion, this Master Thesis presents a working implementation of an annotation system. This system can put forward some strong results and is a significant improvement compared to the baseline. Furthermore, the architecture is designed in such a way that the presented work can serve as a basis for a semantic search engine or other future applications.

6.2 Future work

The objective of this MSc-thesis was to deliver a working system for annotating and storing web pages on events. Next logical step is to continue with the development of the search engine side. This poses some new interesting challenges:

- Experience shows that the success of the design of a user interface for search is highly dependent on and sensitive to the details of the design [28] and the choice between different facet ranking algorithms is at the heart of this problem.
- Social networking sites such as Facebook[1] and Google+[2] are increasingly encapsulating our online identity: users are obligated to provide their real name and a working mail address before they are able to join; by becoming "fan" of something – this can be almost anything such as a person, group, company, music instrument, etc – a list is created of all the user's different interests; they can share information on their wall; a growing number of sites also allow users to authenticate themselves using their Facebook, Google or Twitter profile. Notwithstanding possible security and privacy threats [35, 30], it would be an interesting challenge to build a system that suggest suitable events based on the different interests somebody has listed on his Facebook profile.
- Microblogs such as Twitter[3] only allow for 140 characters. Because of this limitation, users tend to disregard grammatical rules, use a abbreviations, and ommit less important words. These constraints are most likely to have an effect on the quality of the annotations produced by the proposed system. It would make an interesting research topic to measure this effect and compare it with other annotation methods.

[1] www.facebook.com
[2] plus.google.com
[3] www.twitter.com

- Different sources often write about the same events. It would be interesting to see if it is possible to group articles on the same event together based on the produced annotations in these articles. On the same token, events occasionally get cancelled or moved to another date and/or location. A useful feature would be for the system to 'understand' these changes and only present the user with the most updated information (or at least warn the user for possible changes).

Bibliography

[1] P.S. Bayerl, H. Lüngen, U. Gut, and K.I. Paul. Methodology for reliable schema development and evaluation of manual annotations. In *Proceedings of the Workshop on Knowledge Markup and Semantic Annotation at the Second International Conference on Knowledge Capture (K-CAP 2003)*. K-CAP, 2003.

[2] D. Beckett and B. McBride. RDF/XML syntax specification (revised). *W3C recommendation*, 10, 2004.

[3] T. Bemers-Lee, R. Cailliau, A. Luotonen, H.F. Nielsen, and A. Secret. The world wide web. *Communications of the ACM*, 37(8):76–82, 1994.

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

[5] P. Blunsom. Hidden markov models. *Lecture notes, August*, 2004.

[6] A.R.S.S. Board. RSS 2.0 Specification. http://www.rssboard.org/rss-2-0-8, 2007.

[7] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *ACM SIGMOD Record*, volume 30, pages 175–186. ACM, 2001.

[8] B. Bos, T. Çelik, I. Hickson, and H.W. Lie. Cascading Style Sheets, level 2 revision 1 CSS 2.1 Specification. *W3C working draft, W3C, June*, 2005.

[9] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, et al. Extensible markup language (XML) 1.0, 2000.

[10] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0: W3C Candidate Recommendation 27 March 2000. 2000.

[11] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics, 1992.

[12] S. Brin. Extracting patterns and relations from the world wide web. *The World Wide Web and Databases*, pages 172–183, 1999.

[13] K. Idehen C. Bizer, T. Heath and T. Berners-Lee. Linked data on the web (LDOW2008). In *Proceeding of the 17th international conference on World Wide Web*, pages 1265–1266. ACM, 2008.

[14] P. Cimiano, S. Handschuh, and S. Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web*, pages 462–471. ACM, 2004.

[15] D. Ciravegna et al. Adaptive information extraction from text by rule induction and generalisation. 2001.

[16] SEKT Consortium et al. PROTON ontology, 2009.

[17] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, and C. Ursu. The GATE user guide, 2002.

[18] H. Cunningham, D. Maynard, and V. Tablan. JAPE: A java annotation patterns engine. 1999.

[19] M. Dean, G. Schreiber, S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. OWL web ontology language reference. *W3C Recommendation February*, 10, 2004.

[20] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J.A. Tomlin, et al. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th international conference on World Wide Web*, pages 178–186. ACM, 2003.

[21] A. Dingli, F. Ciravegna, and Y. Wilks. Automatic semantic annotation using unsupervised information extraction and integration. In *Proceedings of SemAnnot 2003 Workshop*, 2003.

[22] R.O. Duda, P.E. Hart, and D.G. Stork. Pattern Classification, chapter Nonparametric Techniques, 2000.

[23] R.H.P. Engels and T.C. Lech. Generating ontologies for the semantic web: OntoBuilder. *Towards the Semantic Web*, pages 91–115, 2003.

[24] J. Geng and J. Yang. Autobib: Automatic extraction of bibliographic information on the web. In *Database Engineering and Applications Symposium, 2004. IDEAS'04. Proceedings. International*, pages 193–204. Ieee, 2004.

[25] T. Gruber. What is an ontology? WWW Page, 1992. http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2008.

[26] K. Bontcheva V. Tablan N. Aswani I. Roberts G. Gorrell A. Funk A. Roberts D. Damljanovic T. Heitz M.A. Greenwood H. Saggion J. Petrak Y. Li W. Peters et al H. Cunningham, D. Maynard. *Developing Language Processing Components with GATE*, chapter Performance Evaluation of Language Analysers. The University of Sheffield, Department of Computer Science, 6 edition, 2011.

[27] S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM—semi-automatic creation of metadata. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 165–184, 2002.

[28] M. Hearst. Design recommendations for hierarchical faceted search interfaces. In *ACM SIGIR Workshop on Faceted Search*, pages 1–5, 2006.

[29] M. Hildebrand, J.R. Ossenbruggen, and L. Hardman. An analysis of search-based user interaction on the semantic web. *CWI. Information Systems [INS]*, (E0706), 2007.

[30] H. Jones and H. Soltren. Facebook: Threats to privacy. *Social Science Research*, pages 1–76, 2005.

[31] M. Kifer. Rule interchange format: The framework. *Web Reasoning and Rule Systems*, pages 1–11, 2008.

[32] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):49–79, 2004.

[33] G. Klyne and J.J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. *Changes*, 10(February):1–20, 2004.

[34] P. Kogut and W. Holmes. AeroDAML: Applying information extraction to generate DAML annotations from web pages. In *First International Conference on Knowledge Capture (K-CAP 2001). Workshop on Knowledge Markup and Semantic Annotation, Victoria, BC, Canada*, 2001.

[35] D. Balzarotti L. Bilge, T. Strufe and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *Proceedings of the 18th international conference on World wide web*, pages 551–560. ACM, 2009.

[36] J. Dean L.A. Barroso and U. Holzle. Web search for a planet: The Google cluster architecture. *Micro, IEEE*, 23(2):22–28, 2003.

[37] A. Maedche and S. Staab. Ontology learning for the semantic web. *Intelligent Systems, IEEE*, 16(2):72–79, 2001.

[38] V. Mascardi, V. Cordì, and P. Rosso. A comparison of upper ontologies. In *Atti del Workshop Dagli Oggentti agli Agenti, WOA, M. Baldoni and et al., Eds. Seneca Editore*, pages 55–64, 2007.

[39] L. Masinter, T. Berners-Lee, and R.T. Fielding. Uniform resource identifier (URI): Generic syntax. 2005.

[40] D. Maynard, V. Tablan, C. Ursu, H. Cunningham, and Y. Wilks. Named entity recognition from diverse text types. In *Recent Advances in Natural Language Processing 2001 Conference*, pages 257–274, 2001.

[41] D.L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10:2004–03, 2004.

[42] N.F. Noy and D.L. McGuinness. Ontology developement 101: A guide to creating your first ontology. *Stanford Education*, 2001.

[43] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[44] M. Petrillo and J. Baycroft. Introduction to Manual Annotation, 2010.

[45] E. Prud'Hommeaux and A. Seaborne. SPARQL query language for RDF. *W3C working draft*, 4(January), 2008.

[46] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[47] D. Raggett, A. Le Hors, I. Jacobs, et al. HTML 4.01 Specification. *W3C recommendation*, 24, 1999.

[48] L. Reeve and H. Han. Survey of semantic annotation platforms. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1634–1638. ACM, 2005.

[49] M. Fischetti T. Berners-Lee and Foreword By M.L.Dertouzos. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation, 2000.

[50] M. Tallis. Semantic word processing for content authors. In *Proceedings of the Knowledge Markup & Semantic Annotation Workshop, Florida, USA*, 2003.

[51] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna. MnM: Ontology driven semi-automatic and automatic support for semantic markup. *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 213–221, 2002.