UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGKI
artificial intelligence research koblenz

Distributed Natural Language Search Using Graph-Based Parsing

Masterarbeit
zur Erlangung des Grades
MASTER OF SCIENCE
im Studiengang Informatik

vorgelegt von

Nadine Sina Kurz

Betreuer: Dipl.-Inform. Markus Maron. Betreuer, Arbeitsgruppe Künstliche Intelligenz, Institut für Informatik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Dipl.-Inform. Markus Maron. Betreuer, Arbeitsgruppe Künstliche Intelligenz, Institut für Informatik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Prof. Dr.-Ing. Ulrich Furbach, Arbeitsgruppe Künstliche Intelligenz, Institut für Informatik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Mai 2013

Kurzfassung

Wir präsentieren die konzeptuellen und technologischen Grundlagen einer verteilten natürlichsprachlichen Suchmaschine, die einen graph-basierten Ansatz zum Parsen einer Anfrage verwendet. Das Parsing-Modell, das in dieser Arbeit entwickelt wird, generiert eine semantische Repräsentation einer natürlichsprachlichen Anfrage in einem 3-stufigen, übergangsbasierten Verfahren, das auf probabilistischen Patterns basiert. Die semantische Repräsentation einer natürlichsprachlichen Anfrage wird in Form eines Graphen dargestellt, der Entitäten als Knoten und deren Relationen als Kanten repräsentiert. Die präsentierte Systemarchitektur stellt das Konzept einer natürlichsprachlichen Suchmaschine vor, die sowohl in Bezug auf die einbezogenen Vokabulare, die zum Parsen der Syntax und der Semantik einer eingegebenen Anfrage verwendet werden, als auch in Bezug auf die Wissensquellen, die zur Gewinnung von Suchergebnissen konsultiert werden, unabhängig ist. Diese Funktionalität wird durch die Modularisierung der Systemkomponenten erreicht, die externe Daten durch flexible Module anspricht, welche zur Laufzeit modifiziert werden können. Wir evaluieren die Leistung des Systems indem wir die Genauigkeit des syntaktischen Parsers, die Präzision der gewonnenen Suchergebnisse sowie die Geschwindigkeit des Prototyps testen.

Abstract

We present the conceptual and technological foundations of a distributed natural language interface employing a graph-based parsing approach. The parsing model developed in this thesis generates a semantic representation of a natural language query in a 3-staged, transition-based process using probabilistic patterns. The semantic representation of a natural language query is modeled in terms of a graph, which represents entities as nodes connected by edges representing relations between entities. The presented system architecture provides the concept of a natural language interface that is both independent in terms of the included vocabularies for parsing the syntax and semantics of the input query, as well as the knowledge sources that are consulted for retrieving search results. This functionality is achieved by modularizing the system's components, addressing external data sources by flexible modules which can be modified at runtime. We evaluate the system's performance by testing the accuracy of the syntactic parser, the precision of the retrieved search results as well as the speed of the prototype.

Erklärung

Contents

List of Tables

List of Listings

List of Figures

List of Abbreviations

API	Application Programming Interface
CFG	Context-Free Grammar
DBMS	Database Management System
DM	Data Module
DTD	Document Type Definition
FOL	First-Order Logic
HMM	Hidden Markov Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency
IP	Internet Protocol
IQR	Intermediate Query Representation
JAPE	Java Annotation Patterns Engine
LGD	LinkedGeoData
LOD	Linked Open Data
ML	Maximum Likelihood
NAPA	Navigationsempfänger Chipsatz für Personennavigation mit Anwendungen bei erhöhter Genauigkeit
NER	Named Entity Recognition

NL	Natural Language
NLI	Natural Language Interface
NP	Noun Phrase
OWL	Web Ontology Language
POS	Part-of-Speech
PCFG	Probabilistic Context-Free Grammar
QA	Question Answering
RAM	Random Access Memory
RDF	Resource Description Framework
REST	Representational State Transfer
SGML	Standard Generalized Markup Language
SPARQL	SPARQL Protocol And RDF Query Language
SQL	Structured Query Language
SRL	Semantic Role Labeling
TAG	Tree-Adjoining Grammar
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Vocabulary Module
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

The World Wide Web provides an enormous supply of data that allows to publish and access documents as part of a global information space (6). The development of Semantic Web Technologies has additionally provided a broad variety of methods to store and query knowledge using Web standards and a common data model (7). The availability of structured data increases the necessity for tools bridging the gap between formalized database query languages and the informal natural language of human users (8) (9). To retrieve information from a structured data source, users have to acquire knowledge about the formal query language and the internal organization of its underlying database management system.

Natural Language Interfaces (NLI)s provide a simple and intuitive way for users to enter an informal query and to shift the task of processing the query to the server side in an automated fashion. NLIs facilitate users the access to structured data by taking over the task of generating a database query from a natural language sentence, retrieving query results from the database and presenting the search results to the user in a human-readable form.

A significant benefit of NLIs is their large spectrum of possible areas of application, reaching from general-purpose search engines to highly specified interfaces optimized for specific tasks. With the increasing adoption of web and mobile services offering personalized and location-based services, the possible areas of application for natural language systems has expanded further: NLIs could be employed for querying personal applications such as calendars, contact lists or location-based services. However, the access to customized applications generally requires the employment of highly specified search functionalities, resulting in a high number of possible interfaces. A preferable way of employing a NLI would thus be the integration of multiple data sources in one NLI, providing a simple interface to users while grounding on a large variety of knowledge. Ideally, a distributed NLI would accept any kind of question, automatically recognize what the user is searching for and consult the relevant data source to retrieve an answer.

This thesis presents a framework for a distributed natural language search system, comprising of a novel approach for processing natural language queries with a graph-based parsing model as well as a modularized system architecture that is both independent in terms of the system's vocabularies and knowledge sources. The presented parsing model combines methods of shallow and deep natural language parsing, consisting of a tokenization as well as a 3-staged semantic interpretation of a natural language query using specified patterns.

We develop a modularized system architecture which enables the system to be based on multiple vocabularies for parsing natural language as well as multiple, possibly heterogeneous knowledge sources for information retrieval. Our goal is to develop a natural language interface which integrates both the employment of multiple vocabularies to dynamically adapt the domain of possible queries the system is able to parse, as well as the employment of multiple knowledge sources for extending the scope of possible search results. The system prototype is able to answer general knowledge-based questions as well as location-based questions considering user-specific meta data included in the query parameters. We demonstrate the system's utilization as an integrated module within the mobile application of the pedestrian navigation system NAPA[1] as well as with a web-based interface.

1.2 Structure

This thesis is structured as follows: Chapter 2 provides an overview about the related work of the presented thesis, which consists of the major approaches considering deep and shallow natural language parsing as well as NLI architectures for query generation and possibly distributed information retrieval. We will discuss the concepts of deep parsing as well as challenges of approaches using lexicalized grammars and probabilistic methods. Then we will introduce the major approaches of shallow parsing such as Part-of-Speech Tagging, Chunking, Named Entity Recognition and Semantic Role Labeling. We will then discuss the issue of how to generate database queries for various database management systems from the syntactic parse as well as the issue of how to query multiple, possibly heterogeneous databases. Finally, we will show some approaches considering how to merge and rank search results retrieved from different data sources. In chapter 3 we will define the requirements of our system. The functional requirements address the issue of query parsing and search results retrieval as well as the issue

[1]http://projekt-napa.de

of user interaction. The non-functional requirements focus on the issues of system design, deployment and system interfaces. The developed parsing model will be introduced in chapter 4. It consists of the tasks of identifying semantic tokens within the natural language query (*tokenization*) as well as the generation of a graph-based logical intermediate representation by employing specified patterns (*semantic interpretation*). In chapter 5 we will present the architecture for a distributed natural language search system. The system will be independent both in terms of the underlying vocabulary for parsing natural language as well as for the data sources consulted for finding search results. A prototype of the system will be presented in chapter 6. We will demonstrate the implementation of the foundations presented in chapter 4 and 5, its vocabularies and knowledge sources. The employment of the prototype for users will be demonstrated by integrating the system in the mobile application of the pedestrian navigation system NAPA as well as a browser-based web interface. An evaluation of the system prototype is presented in chapter 7, comprising of the testing of the system accuracy, the system scope in terms of precision and recall of the system, as well as a measurement of the prototype's query processing time. We will then discuss the strengths and weaknesses of the system's components and give an outlook for further development of the system. A summary of the presented approach is given in chapter 8.

Chapter 2

Related Work

This chapter provides an introduction about the theoretical and conceptual foundations of the methods employed by a natural language interface. Based on the architectural designs of a NLI presented by (10) and (4), we divide the scope of a distributed NLI's applied concepts into three stages: 1) The parsing of a natural language query, 2) the knowledge retrieval of various data sources, and 3) the aggregation of the distributed results to a final search response. Within this context, the purpose of syntactic parsing is the analysis of a natural language sentence and the generation of an intermediate representation depicting the natural language's semantics in a logical form able to be processed further by succeeding system components. Though the broad variety of processing approaches, (11) divides natural language parsing into approaches performing a detailed linguistic analyis based on a formal grammar theory (Deep Parsing), and approaches intending to provide lighter, more flexible approaches that often focus on solving a particular task rather than performing a full parse (Shallow Parsing). Section 2.1 will outline the main definitions of Deep Parsing and the concepts of lexicalized and probabilistic parsing. The concept of shallow parsing techniques as well as the main methods Part-of-Speech Tagging, Text Chunking, Named Entity Recognition and Semantic Role Labeling are introduced in Section 2.2. The second aspect considering the concepts of a Question Answering (QA) system is the process of information retrieval from knowledge sources in order to generate answers. We will present approaches to retrieve data from varying data formats such as relational databases and semantic triplestores. Further we will introduce approaches considering distributed data retrieval from multiple knowledge sources. The third aspect is the generation of a distributed system's final result set. Section 2.4 addresses the issue of merging and ranking results from different data sources.

2.1 Deep Parsing

Deep natural language parsing as defined by (11) is characterized by the ambition to apply as much linguistic knowledge as possible to analyze natural language utterances, realizing a detailed syntactic analysis based on a linguistic grammar theory. A distinctive feature of deep parsing methods is the declarative encoding of linguistic knowledge in formal grammars, separating the syntax and semantics of a language from the parsing algorithms (12). The generally rule-based systems describe a language's linguistics abstract from concrete words (11).

Grammars A grammar can be interpreted as a set of transformation rules for generating a language (13), i.e. a set of rules that manipulate symbols (1). A grammar rule consists of terminal symbols that constitute elements of the target language, and non-terminal symbols (or *variables* (14)), which can be interpreted as auxiliary symbols (1). A grammar rule is employed by substituting the left-hand side by the right-hand side of the rule (13), the sequence of substitutions to obtain a string is called a derivation (14). The beginning of the transformation is denoted with a non-terminal S called the start symbol, which is substituted by a word according to the grammar rules (13). As terminals are generally not substituted further, the transformation finishes as soon as a word consists only of terminal symbols (13). A special case of a formal grammar is a set of rules containing a non-terminal symbol V on the left-side of a substitution rule, and an arbitrary sequence of non-terminal and terminal symbols w on the right side. The set of substitution rules thus is of the form $V \to w$, referred to as a *context-free grammar* (13). The definition of a grammar (13), in particular a context-free grammar (1) (14) (12), is read as follows:

Definition 1 *(Context-Free Grammar)*
A context-free grammar is a four-tuple $G = (V, \Sigma, P, S)$*, where*

- V *is a finite set of non-terminal symbols*
- Σ *is an alphabet of terminal symbols*
- $P \subseteq V \times (V \cup \Sigma)^*$ *is a set of rules*
- $S \in V$ *is the start symbol*

The language generated by a grammar can be defined as the set of all terminal words that can be generated from the start symbol S by employment of the grammar's substitution rules (13) (14):

$$L(G) := \{w \in \Sigma^* \mid S \Longrightarrow_G^* w\} \tag{2.1}$$

As it may be possible to apply multiple rules on a word, the generation of a word w' from a word w is an indetermined procedure (13). The employment of a context-free grammar for generating each string of a language is described by (14) as follows:

1. Write down the start variable. It is the variable on the left-hand side of the top rule, unless specified otherwise.
2. Find a variable that is written down and a rule that starts with that variable. Replace the written down variable with the right-hand side of that rule.
3. Repeat step 2 until no variables remain

The sequence of substitutions to obtain a string, called the derivation, is formally defined by (1) as follows:

Definition 2 *(Derivation)*
Let $G = (V, \Sigma, P, S)$ be a grammar. The set of forms induced by G is $(V \cup \Sigma)^*$. A form α immediately derives a form β, denoted by $\alpha \Rightarrow \beta$, if and only if there exist γ_l, $\gamma_r \in (V \cup \Sigma)^*$ such that $\alpha = \gamma_l A \gamma_r$ and $\beta = \gamma_l \gamma_c \gamma_r$, and $A \rightarrow \gamma_c$ is a rule in P. A is called the selected symbol.

A sample set of grammar rules is presented by (1), including a set of terminals {the,cat,in,hat} and a set of non-terminals {D,N,P,NP,PP}:

D → the	NP → D N
N → cat	PP → P NP
N → hat	NP → NP PP
P → in	

Table 2.1: Example of a set of rules comprising a simple grammar (1)

(1) provides an intuitive description of a grammar rule interpretation: If we interpret NP as the syntactic category *noun phrase*, D as *determiner*, and N as *noun*, then what the rule NP → D N informally means is that one possible way to construct a noun phrase is by concatening a determiner with a noun. More generally, a rule specifies one possible way to construct a "phrase" of the category indicated by its head: this way is by concatening phrases of the categories indicated by the elements in the body of the rule (1). A derivation can be visualized by a parse tree consisting of a finite set of vertices connected by a finite set of branches (1). (13) defines the following criteria for a parse tree $B = (W, E, v_0)$ for a context-free grammar:

- Each node $v \in W$ is denoted with a symbol from $V \cup T \cup \{\epsilon\}$

- The root v_0 is denoted with S
- Each inner node is denoted with a variable of V
- Each leaf ist denoted with a symbol of $T \cup \{\epsilon\}$
- If $v \in W$ is an inner node with the child nodes $v_1, ..., v_k$, and v is denoted with A and v_i is denoted with A_i, then $A \rightarrow A_1 ... A_k \in R$
- A leaf denoted with ϵ has no neighboured leaves

A sample parse tree provided by (1) from the grammar shown above is depicted in Figure 2.1.

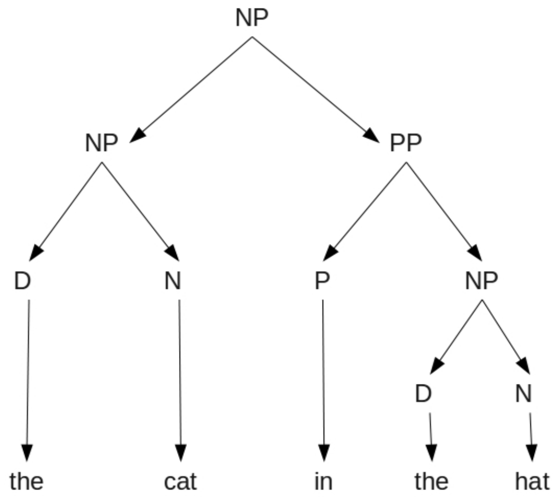


Figure 2.1: Sample parse tree with the grammar presented in Table 2.1 (1)

Lexicalization A major aspect of natural language parsing using grammars is whether a parse is based on individual words rather than on a word's part of speech. A lexicalized grammar realizes syntactic structures that are sensitive to terminal symbols, also called *lexical elements* (12). A common solution is to incorporate a lexical element as a so-called *head* in each non-terminal of the Context-Free Grammar (CFG) (12). (15) defines a grammar formalism as lexicalized if it consists of

- a finite set of structures to be associated with lexical items, which usually will be heads of these structures

- an operation or operations for composing the structures. The finite set of structures define the domain of locality over which constraints are specified and these are local with respect to their lexical heads

(12) describes a model called bilexical context-free grammars, which is a CFG with non-terminal symbols of the form $A[a]$, where a is a terminal symbol and A is a *delexicalized non-terminal.* Every rule in a bilexical context-free grammar has one of the following forms (12):

$$A[a] \rightarrow B[b]\ C[a]$$

$$A[a] \rightarrow B[a]\ C[c]$$

$$A[a] \rightarrow B[a]$$

$$A[a] \rightarrow a$$

A general parsing strategy for Tree-Adjoining Grammar (TAG)s based on lexicalized grammars is presented by (15). (16) proposes a statistical parsing model using lexicalized context-free grammars. A general parsing strategy with lecialized grammars applied to TAGs is presented by (15). The benefits of lexicalization have been discussed in various works. An approach of an unlexicalized parser that shows a significantly well performance is presented by (17). (17) points out that unlexicalized Probabilistic Context-Free Grammar (PCFG) parsers are much simpler to build and optimize. On the other hand, (2) argues that lexicalized parsers achieve significantly higher precision-recall accuracies up to 87-percent to 88-percent precision-recall. However, (2) also points out the sparse data problems occuring when gathering statistics on individual words, such as the occurrence of new words or new word combinations where no data is collected yet.

Probabilistic Parsing A common problem in natural language processing is the syntactic ambiguity of terms, i.e. words or sentences with two or more possible meanings. A natural language parser thus is required to be able to compute the most adequate parse with respect to the term's context. A possible solution to ambiguity is the introduction of probabilistic methods in natural language parsing. Calculating a probabilistic value for each possibility, a probabilistic parser may choose the most likely parse on the base of a heuristic.

A simple way to add probability to a formal grammar is to associate each grammar rule with a probability (12). A definition of a PCFG is presented by (12):

Definition 3 *(Probabilistic Context-Free Grammar)*
A probabilistic context-free grammar is of the form $G = (V, \Sigma, P, S, p)$, *where*

(V, Σ, P, S) *is a context-free grammar and* p *is a mapping from rules in* P *to be real numbers between 0 and 1.*

(12) defines a PCFG as *proper* if for every non-terminal A, p defines a probability distribution over the rules with left-hand side A, i.e.

$$\sum_{A \to \alpha} p(A \to \alpha) = 1 \tag{2.2}$$

(18) describes the statistical parsing model as follows: The model defines a conditional probability $P(T|S)$ for each candidate parse tree T for a sentence S. The parser itself is an algorithm which searches for the tree T_{best}, that maximises $P(T|S)$. The probability of a parse is calculated as the product of the probabilities for each of the rules used therein (2) (12). If s is the entire sentence, π is a particular parse of s, c ranges over the constituents of π, and $r(c)$ is the rule used to expand c, then

$$p(s, \pi) = \prod_c p(r(c)). \tag{2.3}$$

Probabilistic approaches have been presented by a variety of works (19) (20).

2.2 Shallow Parsing

Other than deep parsing, shallow parsing techniques as described by (11) (21) intend to produce a more lightweight, flexible representation of a natural language input and often focus on performing a particular task rather than a full parse. Shallow parsing techniques generally segment a text into logical semantic units and retrieve their semantic roles within the sentence's context.

This section will provide an overview of the most important techniques of shallow natural language parsing. First we will introduce Part-of-Speech tagging, which describes the disambiguation of words within a sentence (22). This analysis is extended by text chunking, which focuses on the identification of logical, non-overlapping groups of words in a text (*chunks*) (23). A more differentiated recognition is provided by Named Entity Recognition (NER), which recognizes specific instances within a text. A defined role of a logical unit is identified by semantic role labeling.

2.2.1 Part-Of-Speech Tagging

Part-of-Speech (POS) tagging describes the identification of a word's syntactic category within the context of its sentence such as noun, pronoun, verb, adjective or adverb (24) (11). A POS tagger separates a natural language sentence into segments and enriches the corpus by the semantic roles of each word. A major challenge of POS tagging is lexical disambiguation: (25) points out that, depending on the context, e.g. the word *"store"* can be either a noun, a finite verb or an infinitive. An example of the possible parts of speech of the words in a sentence is presented by (2):

The	can	will	rust
det	modal-verb	**modal-verb**	noun
	noun	noun	**verb**
	verb	verb	

Table 2.2: Lexical disambiguation by POS tagging (2)

The problem of tagging has been described by (26) as follows: Consider a sentence consisting of a set of words $W = w_1w_2...w_n$, and a sequence of tags $T = t_1t_2...t_n$, of the same length. The pair (W,T) constitutes an alignment, where a word w_i has been assigned the tag t_i. Therefore, a tagging procedure is a procedure ϕ which selects a sequence of tags (and so defines an alignment) for each sentence.

$$\phi : W \rightarrow T = \phi(W) \tag{2.4}$$

POS tagging has been the field of research of various works. An introduction to POS tagging and partial parsing is given by (22). An approach to apply transformation-based error-driven learning to POS tagging is presented by (27). POS tagging approaches can generally be divided into rule-based and statistical systems.

Rule-based POS tagging According to (24), rule-based POS approaches assign tags to words based on a lexicon and a set of hand-crafted or learned rules. Early approaches employing taggers with hand-constructed rules were presented by (28) (29). To facilitate the process of acquiring rules for POS tagging, various techniques for acquiring rules automatically have been developed. (30) presents a procedure for automatically acquiring a set of disambiguation rules for an existing deterministic parser on the basis of tagged text. A simple disambiguation rule presented by (30) looks like this:

$$[PREP\ +\ TNS] = TNS\ [N + V]$$

indicating that a word that can be a preposition or a tense marker (i.e. the word *to*) followed by a word which can be a noun or a verb is a tense marker followed by a verb (30).

(31) presents an approach which automatically acquires its rules and tags. As advantages of rule-based over stochastic POS taggers (31) names a vast reduction in stored information and a better portability. An approach inspired by (31) that implements a finite-state tagger is presented by (32). On the other hand, (22) points out the amount of effort necessary to write the disambiguation rules of rule-based POS taggers.

Statistical POS tagging Statistical tagging approaches use a variety of probabilistic techniques in order to assign POS tags to unseen text. (26) describes a probabilistic formulation of the tagging problem as alignments generated by a probabilistic model according to a probability distribution:

$$p(W,T) \tag{2.5}$$

Depending on the criterion chosed for evaluation, (26) formulates the optimal tagging procedure:

- for evaluation at sentence level: Choose the most probable sequence of tags for the sentence (Viterbi tagging)

$$\phi(W) = \arg\max_T p(T/W) = \arg\max_T p(W,T)$$

- for evaluation at word level: Choose the most probable tag for each word in the sentence (Maximum Likelihood (ML) tagging)

$$\phi(W)_i = \arg\max_t p(t_i = t/W) = \arg\max_t \sum_{T:t_i=t} p(W,T)$$

(2) describes a statistical model of POS tagging as follows: The most common tag t for the ith word of a sentence w_i, is the one that maximizes the probability $p(t|w_i)$, that is, by finding the t that maximizes the probability of a tag given the word.

$$\arg\max_t p(t|w_i) \tag{2.6}$$

Extended to an entire text, (2) describes that the parser looks for the sequence of n tags $t_{1,n}$ that maximizes the product of the individual word probabilities

$$\arg\max_{t_{1,n}} \prod_{i=1}^{n} p(t_i|w_i) \tag{2.7}$$

Although (2) points out the significant accuracy of this algorithm, it does not consider a word's context yet. A possible approach for context-specific probabilistic tagging is proposed by (2) by collecting statistics on the probability of tag t_{i-1}

$$\arg\max_{t_{1,n}} \prod_i p(t_i|t_{i\pm1})p(w_i|t_i) \tag{2.8}$$

In this method (2) takes into account two probabilities: 1) the probability $p(t_i|t_{i-1})$ of a tag t_i given the previous tag t_{i-1} as context and 2) the probability of a word's possible tags $p(w_i|t_i)$.

Probabilistic approaches using Markov models have been presented by (26) (33). A probabilistic POS tagging approach using decision trees is proposed by (25). (34) introduces a memory-based approach to POS tagging, where the POS tag of a word in a particular context is extrapolated from the most similar cases held in memory. A statistical model using a maximum entropy model for POS tagging is presented by (35).

An evaluation of various POS tagging approaches is performed by (36). POS tagging provides a first classification of a sentence's words and is thus implemented as a preceding step of an extensive analysis by various works (37).

2.2.2 Chunking

Text chunking as defined by (23) describes the dividing of text into syntactically related non-overlapping groups of words. (38) describes that a typical chunk consists of a single content word sourrounded by a constellation of function words, matching a fixed template. In this context, (38) defines chunks in terms of *major heads*, that is, all content words except those that appear between a function word f and the content word that f selects. As an illustrative example, (38) points out that the sentence *a man proud of his son* contains *proud* as a major head, while it is not a major head in *the proud man.* As most chunks consist of multiple words, (38) points out that lexical ambiguity is often resolvable within chunks. In this context, text chunking provides a wider spectrum than POS tagging. Various works consider POS tagging as a part of chunking (39), if it is assumed each character as a token.

An example of a non-overlapping sentence segmentation is presented by (23), where chunks are represented as groups of words between square brackets and a tag next to the open bracket denoting the type of the chunk:

[$_{NP}$ He] [$_{VP}$ reckons] [$_{NP}$ the current account deficit] [$_{VP}$ will narrow]
[$_{PP}$ to] [$_{NP}$ only £ 1.8 billion] [$_{PP}$ in] [$_{NP}$ September] .

Considering the types of chunks, (40) distinguishes between noun phrases(NP), verb phrases(VP), adverbs (ADVP) and adjectives (ADJP), prepositions (PP) and clauses introduced by a subordinating conjunction (SBAR) as well as conjunctions (CONJP), verb particles (PRT), interjection phrases (INTJ), list markers (LST) and unlike coordinated phrases (UCP).

Machine learning has been applied to chunking by (41), who show that it becomes possible to easily apply transformation-based learning by representing text chunking as a kind of tagging problem. They employ a chunk tag set I, O, B, where words marked I are inside some non-recursive base noun phrase, those marked O are outside, and the B tag is used to mark the left most item of a base noun phrase which immediately follows another baseNP.

(42) presents an approach for memory-based shallow parsing techniques to find labeled chunks and grammatical relations in a sentence. An approach for memory-based learning to fast Noun Phrase (NP) chunking is presented by (43). (39) introduces a framework for chunking based on Support Vector Machines.

2.2.3 Named Entity Recognition

NER is the task of classifying nouns in a document in one of a defined set of possible categories, where (44) names persons, organizations and locations as possible categories. Other than other shallow techniques and deep parsing techniques analyzing the input sentence's structure, NER concentrates on concrete words and the association of nouns with specific entities. While POS tagging and chunking are focused on identifying the syntactic role of one or multiple words, NER concentrates on identifying the concrete role of a noun. For example, (45) uses the following categories: person, location, organization, date, time,percentage, monetary value, and "none-of-the-above". NER is generally performed by marking a sentence with Standard Generalized Markup Language (SGML) tags. According to (45) (46), the sample sentence "X is an analyst who has been in Koblenz since 2010" would therefore be marked with SGML tags as follows:

```
<ENAMEX TYPE='PERSON'>X</ENAMEX>
is an analyst who has been in
<ENAMEX TYPE='LOCATION'>Koblenz</ENAMEX> since
<TIMEX TYPE='DATE'>2010</TIMEX>.
```

While the intuition of POS tagging is to define the role of each word in the input string in terms of its syntactic role within the sentence's context, NER identifies single instances and the categories they belong to. Statistical NERs are described by (47) as finding the sequence of tags that maximizes the probability $p(N|S)$, where S is the sequence of words in a sentence, and N is the sequence of named-entity tags assigned to the words in S. Several techniques have been applied to NER, such as Maximum Entropy (47) (45), Hidden Markov models (48) and Conditional Random Fields (49).

To retrieve the type of a noun, most NER systems employ large lists of names of people, organizations, locations etc. However, (46) presents an approach for NER with relatively small gazetteers combining rule-based grammars with statistical models that achieves good performance. (50) present a NER recognition system using an Hidden Markov Model (HMM)-based chunk tagger. An approach using the Wikipedia[1] corpus for NER is presented by (51). A detailed analysis of existing NER approaches is found in (52).

2.2.4 Semantic Role Labeling

Semantic Role Labeling (SRL) as defined by (53) describes the task of analyzing a sentence's target verbs and its constituents in the sentence which fill a semantic role. Semantic roles are a set of defined syntactic roles, which can range from the very specific to the very general (54). (53) defines the structure of the arguments in a proposition as non-overlapping and sequentially organized. (53) explains how an utterance argument ($A1$) can be split into two phrases by the following example:

"$[_{A1}$The apple], said John, $[_{C-A1}$is on the table]"

Other than NER, semantic role labeling does not focus on nouns, but rather seeks to identify coherent syntactic structures within a sentence and label them with possibly customized roles. A variety of works using different techniques has been focused on SRL, such as probabilistic models (54), generative models (55) or Support Vector Machines (56).

Section 2.1 and 2.2 presented an introduction of the main approaches of deep linguistic systems as well as methods of shallow syntactic analyses. An analysis of the efficiency of shallow parsing has been performed by (21), whose experiments have shown that a shallow parser trained on specific tasks may perform more accurately and more robustly than a full parser, in particular in handling ill-formed real-world sentences. The distinction between deep and shallow parsing does not mean that both approaches have to be implemented separately. For example, shallow methods can be employed as a preceding step for providing first approximate results for a subsequent detailed syntactic analysis. (11) points out that that the integration of both approaches is also economically motivated, mainly driven by the expensive, time-consuming grammar development of deep systems. Therefore, (11) proposes the following task sharing: Domain-specific extensions could be con-

[1]http://wikipedia.org

tributed by shallow systems, while the core linguistic and grammar theory-based knowledge system is performed by deep parsing. A logic-based question answering system employing an automated theorem prover to infer correct replies is presented by (57). Beside an analysis of a natural language sentence based on deep question parsing, the approach also employs shallow feature extraction and reranking. The semantic representation of a natural language sentence is transformed into a logical query comprising of a conjunctive list of query literals. The approach has further been expanded by a machine learning solution for avoiding wrong answers (58).

2.3 Data Retrieval

After parsing a natural language query and generating an intermediate query representation, a natural language interface retrieves a search response by computing one or multiple queries for consulting the system's knowledge base. This section will address the issue of data retrieval for computing search responses for natural language interfaces. In order to consult a knowledge base, NLIs have to generate a query from the semantic representation of the natural language input and process the results retrieved by the knowledge base. NLIs thus require knowledge about the database's internal organization as well as a formalism to map the logical representation of a natural language input into a valid database query. This section will introduce the challenges and the main approaches for generating database queries from natural language. We will address the issue of generating queries by systems relying on relational databases as well as Resource Description Framework (RDF) triplestores. We will further outline the concept of distributed NLIs, which retrieve data from multiple, possibly heterogeneous knowledge sources.

2.3.1 Natural Language Interfaces to Databases

As (4) outlines, most early approaches to NLIs have been based on relational databases generating Structured Query Language (SQL) queries from a natural language sentence, with the first NLIs developed in the late sixties and early seventies. NLIs consulting knowledge sources based on relational databases need to transform the syntactic parse into an appropriate SQL query. For this purpose the system requires to have information about the databases's internal schema as well as a mechanism how to map elements from the Intermediate Query Representation (IQR) into a valid database query. An overview of NLI approaches relying on databases is presented by (4), including the proposal of a possible architecture of a NLI to databases, depicted in Figure 2.2. As a NLIs main components, (4) proposes a parser and a semantic interpreter for generating an intermediate representation of the query, a database query generator for consulting a database and a response generator.

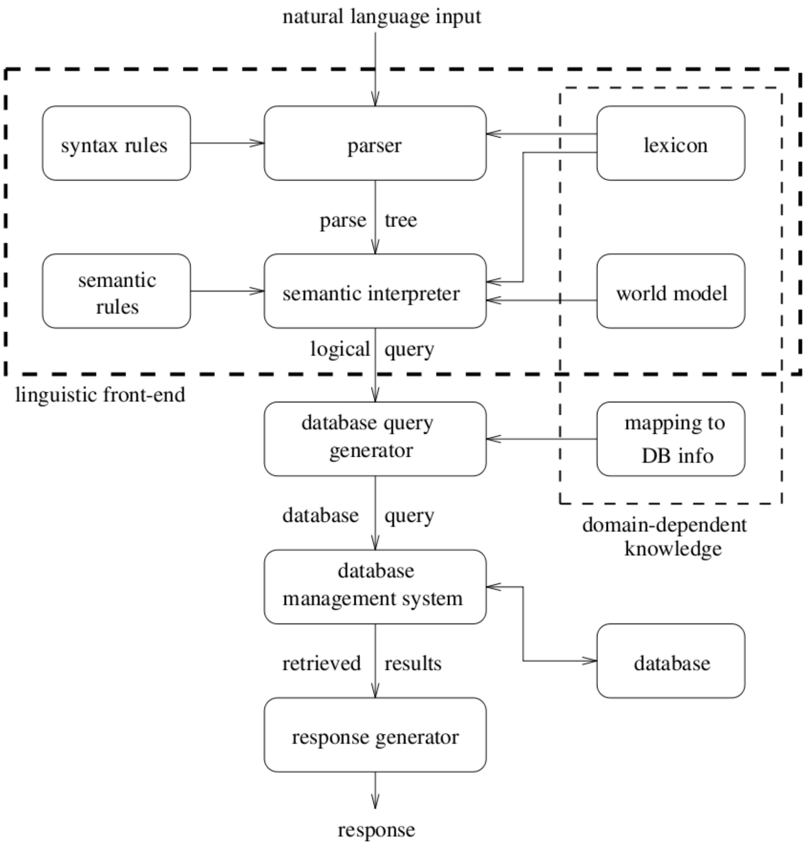


Figure 2.2: Sample architecture of a NLI to databases presented by (4)

One of the major challenges of the automated generation of valid SQL queries is the correct association of fragments of the natural language query with elements of the database schema. (37) points out the `SELECT`, `FROM` and `WHERE` clause as a SQL query's main constituents, which have to be associated with elements of the natural language query. For generating the conditions of the `WHERE` statement, (37) proposes a shortest path approach that associates elements of the natural language query with relations or fields of the database structure.

(59) introduces a NLI to databases with defining a notion of semantically tractable questions the system is able to parse. The approach employs a lexicon mapping stemmed natural language terms to database elements such as database tables, attributes and concrete values. The problem of mapping between words

and tokens such that attributes are connected to their values is formulated in terms of an attribute-value graph.

(60) presents a NLI to relational databases adopting a phrasal approach, i.e. uses a phrasal lexicon rather than a general purpose grammar for syntactic analysis. (61) proposes natural language annotations as a mechanism by which questions are matched to candidate answers. A NLI to relational databases that works by obtaining knowledge automatically from a database and training corpus is presented by (37). NLIs consulting Extensible Markup Language (XML) databases were presented by (62) (63), which translate a parse tree into XQuery expressions. Approches using files of frequently asked questions as its knowledge base have been proposed by (64) (65).

2.3.2 Natural Language Interfaces to RDF

The development of Semantic Web Technologies has enhanced the possibilities of natural language interfaces significantly in terms of domain independency and knowledge retrieval. The terminological knowledge stored in ontologies provides the possibility to publish conceptual data about knowledge domains in a structured way that can be parsed automatically. RDF [2] (66) provides a framework to express data in triples following a subject-predicate-object model that uses the XML encoding as its interchange syntax as well as the XML namespace facility (66). By storing data in the RDF triple format, clients of triplestores do not require additional knowledge about the database's internal organization in order to generate valid queries. Various works have thus employed triple-based data models as the base for the intermediate representation of natural language parses (67) (68). A query language for RDF repositories has been developed with the SPARQL Protocol And RDF Query Language (SPARQL)[3] (69), which allows to formulate graph-based queries.

The emergence of Linked Data, described by (6) as a set of best practices for publishing and interlinking structured data on the Web, has further supported the distribution of Semantic Web Technologies. As many Linked Open Data (LOD) providers have made their data publicly available by providing SPARQL endpoints, the scope of possible applications of RDF-based NLIs has increased significantly.

Various approaches have presented solutions for generating SPARQL queries from natural language. (68) presents an approach of translating parse trees to SPARQL queries by mapping nominal-phrase pairs connected by prepositions or verb phrases to triples of the form <*subject, predicate, object*>. The approach uses a lexicon containing ontology entities including classes (concepts), properties

[2]http://www.w3.org/RDF
[3]http://www.w3.org/TR/rdf-sparql-query

(relations) and instances (individuals) to map words in natural language queries to entities in the ontology. As (68) assumes a triple to represent a semantic relationship between two nominal phrases, a triple is generated by mapping two nominal phrases to a triple in the ontology.

A modular QA system with respect to the input ontology is proposed by (67). The system is based on shallow parsing techniques and handcrafted grammars to identify terms and relations using the GATE[4] (70) framework including the Java Annotation Patterns Engine (JAPE) (3). The employed JAPE grammars of GATE consist of a set of phases, each of which consists of a set of pattern rules, which act on annotations assigned in earlier phases. A JAPE grammar consists of a left-hand side performing pattern matching, and a right-hand side describing the annotation to be assigned (3). A sample JAPE rule identifying an Internet Protocol (IP) address presented by (3) is shown in Listing 2.1.

```
Rule: IPAddress
{
 (
  {Token.kind == number}
  {Token.string == "."}
  {Token.kind == number}
  {Token.string == "."}
  {Token.kind == number}
  {Token.string == "."}
  {Token.kind == number}
 )
 :ipAddress -->
  :ipAddress.Address = {kind = "ipAddress"}
}
```

Listing 2.1: Sample rule of a JAPE grammar (3)

(71) points out that most approaches using triple-based schemas capture simple relationships very well, yet the generation of SPARQL filters such as numeric constraints (e.g. *"Which city has more than three universities?"*), quantifiers (*"How many.. ?"*) or aggregation functions often depict a crucial issue. (71) thus presents an approach that relies on a parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question. An ontology-driven approach that is able to resolve quantifiers and number restrictions is further presented by (72). (73) presents a domain-independent approach generating an intermediate representation in First-Order Logic (FOL), which can be translated into F-Logic and SPARQL queries. Further approaches generating SPARQL queries from natural language have been presented by (74) (75) (76) (77) (78).

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

2.3.3 Distributed Natural Language Interfaces

A promising development for new NLIs is the consultation of multiple knowledge sources for providing domain-independency. Systems querying multiple data sources provide a wide variety of knowledge and could easily be reconfigured to query new knowledge bases. On the other hand, a system relying on multiple data sources requires a mechanism to generate multiple queries from a natural language parse. While early NLIs were mainly designed to parse natural language with a predefined domain and a particular Database Management System (DBMS), the issue of system portability has increasingly gained attention (4). An increased system flexibility enables the system to work with different knowledge domains or different knowledge sources for information retrieval. (4) distinguishes four kinds of portability:

- **Knowledge-domain portability**: Domain-independent systems are able to be easily adapted to new knowledge domains, enabling to understand new words and concepts
- **DBMS portability**: NLIs that can be modified to be used with different database managements systems have a broad range of possible knowledge retrieval for the generation of search responses
- **Natural language portability**: Most NLIs developed so far assume Natural Language (NL) requests to be written in English (4). The adaptation of NLIs to a variety of different languages increases the possible areas of applications. However, (4) points out that this step also typically requires the consideration of different lexica, syntax and semantic rules.
- **Hardware and programming language portability**: The vast change of hardware and software features increases the need for the ability to adapt a NLI to new systems

A promising approach for natural language interfaces is the gathering of data from multiple, possibly heterogeneous, data sources. A system that is able to consult a variety of databases provides a wider spectrum of knowledge and flexibility. However, this approach also requires a more complex architecture: Since the system needs to generate several queries from a natural language parse, the NL parse needs to be stored in a logical intermediate representation, which allows the generation of multiple database queries in the corresponding database query languages as well as an integration of the results retrieved by various database management systems.

An extension of the system presented by (67) towards aggregating information derived from multiple heterogeneous data sources is presented by (79). The system proposed by (80) integrates heterogeneous data sources using an object-property-value-model. An architecture for a collaborative QA system that enables users to contribute to the knowledge base is proposed by (81). The system uses natural language annotations, which are machine-parsable sentences or phrases that describe the content of various information segments. (82) present an approach for a transportable NLI that can be easily adapted to new domains or databases. Most NLI approaches assume data to be stored locally, which ensures data availability and the absence of the necessity to adapt the system to external changes. However, an approach with locally stored data also implies the system's domain to be limited to the available data in the database. The increasing availability of structured data on the World Wide Web such as the provision of SPARQL endpoints has driven new approaches of gathering data on the web at runtime. In this way a NLI's available knowledge base can be significantly increased, further (83) points out that the data will constantly be kept up to date with the web itself. On the other hand, consulting data sources on the web for question answering bears new issues to deal with. (83) names some specific challenges of collecting data on the web:

- **Forming the right queries:** A query has eventually be adapted to the remote data source. Furthermore, the scope of a query has to be considered: If the query is formulated too general, too many results that are unrelated to the query are found. If it is too specific, very few or no results may be retrieved.
- **Noise:** With a growing amount of available data, the number of results that deal with something else increases as well
- **Factoids:** Data on the web often contains invalid data the NLI has to deal with
- **Resource limitations:** (83) points out that the user's willingness to wait a long time for an answer is very limited. A NLI therefore has to be able to answer user queries in an appropriate time

(83) presents a QA system consulting search engines for data retrieval by converting the question into a set of keyword queries that are sent to various search engines for parallel evaluation.

Controlled Natural Language To ensure that a natural language system is able to generate a database query from a natural language sentence, some approaches have restricted the possible input sentences and require the queries to have a specific syntax in order to be parsed by the system. (84) presents an approach using controlled natural language which has a restricted grammar and lexicon. A guided input NLI presenting possible completions of what the user enters and presenting the user with a choice popup box is presented by (85). By restricting the user input sentences, controlled natural language systems make sure to be able to produce a valid parse for the query. However, (4) points out that the limits of the natural language subset are usually not obvious to the user, causing many questions never tried because users think the questions are outside the subset of natural language supported by the NLI.

Dialog Systems NLIs commmonly face the issue of how to process queries that cannot be parsed unambigously. To solve the common problem of choosing one of multiple possible NL query interpretations, a variety of NLIs have implemented user clarification dialogs in order to ask the user to choose the most appropriate interpretation (76) (78) (60) (59).

This section has provided an introduction to various approaches of NLIs to access knowledge sources. For further reading, (10) as well as (86) present overviews of existing approaches to question answering.

2.4 Result Processing

After retrieving data from one or multiple knowledge sources, NLIs process the retrieved results in order to generate a response that is returned to the client. The issues of result processing involve the tasks of merging result sets from different data sources as well as the ranking generation for result lists.

Instance Matching Applications retrieving data from multiple knowledge sources commonly face the problem of identifying redundant results. If we compare the analysis of result instances to the task of ontology matching, we can identify parallels in terms of identifying similar or identical elements within structured data. Various works that have addressed the issue of ontology matching have developed solutions for identifying similarities between ontology classes: An approach for automated ontology merging and alignment is presented by (87). (88) proposes an architecture for discovering ontological concepts and relations, which includes an association rule algorithm detecting relations between concepts.

Ranking Search interfaces that return a list of results require algorithms for sorting the retrieved items in an approriate way. In order to present the most relevant results of a query to the user, a search interface orders the results according to a heuristic and selects the most important results to the user. Ranking algorithms in hyperlinked environments have been presented by (89) (90), considering the links included in web documents as quality indicators. (89) developed a global ranking system for documents on the Web based on their link structures. (90) has proposes the concept of hubs and authorities of web pages and the discovery of authoritative information sources.

An approach for ranking results retrieved from search engines is presented by (83): The approach prefers summaries that contain more important keywords which are close to each other. As a measure for the importance of a keyword they use the *Inverse Document Frequency (IDF)*, which is defined by $\frac{N}{df}$, where N is the size of a document collection and *df* is the number of documents containing the word. For calculating the square-root-mean of the distances between keywords, (83) calculate the distance between the distances $d_1, ..., d_{n-1}$ and n keywords in summary s as

$$D(s) = \frac{\sqrt{d_1 + ... + d_{n-1}}}{(n-1)} \tag{2.9}$$

If s has n keywords, each with weight w_i, then (83) calculates its score as follows:

$$S(s) = \frac{\sum\limits_{i=1}^{n} w_i}{D(s)} \tag{2.10}$$

Chapter 3

Requirements

The development of a software-based system requires the specification of its functionality as well as its conceptual and technological essentials. Based on the software requirements analysis presented by (91), we will divide the system's fundamentals into functional and non-functional requirements.

3.1 Functional Requirements

This section will define the functional requirements for the query processing of a natural language sentence. The steps performed by the QA system from receiving a natural language query to retrieving a list of search results can be divided into three parts: The first is the syntactic parsing of the natural language input and the generation of an intermediate data structure of the parse. The second is the retrieval of data from the system's knowledge sources in order to generate a response. Finally, the results retrieved from the different data sources are merged and ranked according to a specified heuristic and a list of search results is generated. Each requirement will be marked with **R**, where a plus sign indicates a necessary requirement and a minus sign indicates an optional requirement.

3.1.1 Syntactic Parsing

The syntactic parser receives a natural language request from the client, processes the natural language input and generates a logical intermediate representation.

$\mathbf{R}^+$ 1 The syntactic parser receives a natural language query from the system's interface

$\mathbf{R}^+$ 2 In order to parse a natural language query, the syntactic parser proceeds the following steps:

- Structure the query into logical units (*tokens*)
- Generate a graph-based representation of the natural language query depicting relations between entities as nodes connected by edges (*semantic interpretation*)
- Transform the graph into a logical intermediate query representation

Tokenization The first step of the parsing model is the identification of semantic logical units within the natural language sentence. It consists of several syntactical preparations, the segmentation of the query into words or sets of words as well as the identification of uniquely identified resources within these segmentations. In natural language processing, words are often received in different forms, such as plurals of nouns or verbs' tenses. The parser therefore has to compute the base forms of nouns and verbs.

R$^+$ 3 Prepare the natural language sentence for the tokenization

- **R$^+$ 3.1** Remove punctuation marks from the sentence
- **R$^+$ 3.2** Turn the sentence to lower case
- **R$^+$ 3.3** Remove stopwords (on,a,the,etc.) from the sentence
- **R$^-$ 3.4** Remove plurals from nouns
- **R$^-$ 3.5** Generate the present form of verbs

We assume a natural language sentence contains references to semantic entities which can be identified by associating words or sets of words with them. In order to identify coherent words refering to an entity, we will segment the sentence into all possible word combinations. The next step is the identification of semantic entities within the query. As a reference to an entity can possibly consist of multiple words (e.g. *"New Zealand"*), the system has to consider all possible word combinations of the query.

R$^+$ 4 Compute all possible segmentations of the input sentence

- **R$^+$ 4.1** Divide the sentence per whitespaces into words
- **R$^+$ 4.2** Compute all possible word combinations representing how the sentence can be logically segmented

R$^+$ 5 Identify semantic tokens in the computed segmentations

- **R$^+$ 5.1** Identify resources by looking up each word or word combination in a lexicon that associates words with resources on the Web
- **R$^+$ 5.2** A term can be associated with
  - a class of an ontology
  - a property of an ontology

- an instance of a class
- a query constraint

R$^-$ 6 A query constraint can be

R$^-$ 6.1 a numeric restriction

R$^-$ 6.2 a geospatial restriction indicating a certain area for a location-based search

R$^-$ 6.3 a date or a timeframe

R$^+$ 7 If one or multiple associations for a word or a word combination have been found, select the most probable association

R$^-$ 8 Identify query constraints using regular expressions

R$^+$ 9 Identify the most probable segmentation of a natural language query into semantic tokens

R$^+$ 9.1 Select the segmentation that associates the highest fraction of the input sentence with semantic entities

R$^+$ 9.2 If multiple segmentations associate the same fraction of the sentence with semantic entities, choose the segmentation that realizes the association with fewer entities, i.e. provides the simpler solution

An additional feature is the recognition of the geospatial coordinates of entities contained in a natural language query. For example, the query *"universities in Koblenz"* searches for entities located in a specific area. A location-based natural language search thus could identify the geographic coordinates of the point of interest contained in the query and restrict the query to the requested area. Furthermore, an interesting feature of natural language search in location-based applications is the retrieval of location-based related terms with respect to the input query. Search engines could suggest similar points of interest the users might be interested in. In case of product searches the search engine could provide suggestions of locations where the required item could be acquired.

R$^-$ 10 Identify geospatial coordinates of tokens contained in the natural language query

R$^-$ 11 If geospatial coordinates have been identified for a token contained in the natural language query and no geospatial coordinates have been transmitted representing the query's geographic origin within the query's meta parameters, set the token coordinates as the query's geographic origin

R$^+$ 12 Associate products or real-world entities with local businesses where the requested entity can be acquired

Semantic Interpretation The major part of the parsing procedure is the generation of a logical representation of the natural language query, which is achieved by employing a transition-based model using probabilistic patterns. The goal of the parsing model developed in this thesis is the representation of a natural language query in terms of a graph-based representation.

R$^+$ 13 Employ hand-crafted graph patterns to identify entities and relationships within the semantic tokens identified in the previous step

R$^+$ 14 Employ graph patterns by computing the pattern that is most similar to the current stage of the intermediate representation of the natural language parse and apply the pattern's transformation rule

R$^+$ 15 Generate the nodes of the graph, representing the query's entities

- **R$^+$ 15.1** Identify classes, properties, instances and query constraints as nodes
- **R$^+$ 15.2** Identify each node with a Uniform Resource Identifier (URI)
- **R$^+$ 15.3** Identify each node with a probability
- **R$^+$ 15.4** Identify sets of undefined entities described by its linkage within the graph as non-terminal nodes

R$^+$ 16 Generate the relations of the graph, representing query constraints of query entities or relationships between entities

- **R$^+$ 16.1** A relation connects two nodes with a labeled edge
- **R$^+$ 16.2** Identify a relation with a URI
- **R$^+$ 16.3** Identify a relation with a probability
- **R$^+$ 16.4** Identify an undefined relation between two nodes as a non-terminal branch

R$^+$ 17 Identify the focus of the graph, representing the query's main information request

- **R$^+$ 17.1** Identify a node or a branch of the generated semantic graph as the graph's focus
- **R$^+$ 17.2** Each semantic graph has exactly one focus

Query Scope An important aspect of NLIs is the specification of the possible queries the system shall be able to answer. We will consider the broad variety of possible questions in terms of a RDF graph consisting of nodes and edges, which describes a resource with a set of edges connecting it with other resources or literals. Thus we can divide the set of possible natural language queries into searches for nodes, i.e. for a set of resources (*entities*), or into searches for edges, i.e. the value of a property of a specific resource (*fact*). For example, an input query *"Name all mountains above 4000 m"* would search for resources (*mountains*) with a specific constraint ($> 4000m$), embodying an *entity search*. On the other hand, the question *"How high is the Matterhorn?"* would search for a property (*height*) of a specific resource (*fact search*).

Another issue is the parsing of noisy queries. NLIs are generally designed for parsing complex, yet well-structured and grammatically correct sentences. However, as NLIs are increasingly included into frequently used web and mobile applications, users are less willing to enter full sentences and rather expect NLIs to understand informal input such as keywords or short phrases. We thus design our system to be able to cope with both formal sentences as well as informal, possibly noisy queries.

R^+ 18 The system classifies a natural language query according to the intended type of the response

- **R^+ 18.1** Users can search for general entities related to a resource
- **R^+ 18.2** Users can search for general facts of an entity
- **R^+ 18.3** Users can search for location-based entities in their close proximity
- **R^+ 18.4** Users can search for points of interests or local businesses where a certain product or entity can be found or acquired
- **R^+ 18.5** Users can search for location-based entities described by constraints or time intervals

R^+ 19 The system is able to parse natural language queries in multiple languages

- **R^+ 19.1** Users can ask queries to the system in German
- **R^+ 19.2** Users can ask queries to the system in English

R^+ 20 The system is able to parse informal or grammatically incorrect questions

- **R^+ 20.1** The system is able to parse sentences with incorrect word order
- **R^+ 20.2** The system can parse keywords, i.e. a list of nouns

3.1.2 Data Retrieval

After parsing the natural language query into an intermediate representation, the system consults various knowledge sources in order to retrieve search results. One of the major aspects of the system developed in this thesis is the ability to gather data from multiple data sources with possibly different underlying database management systems. The system has one or multiple modules at its disposal that are able to consult a knowledge source each. Knowledge sources may be activated and deactivated dynamically. A crucial aspect of any search interface is furthermore the time the system needs to produce an answer. As users are generally unwillingly to wait a long time for a response, the system should ideally provide an answer within few seconds. This aspect is especially important for NLIs collecting data on the web from different sources, as they have no control about the time the consulted web data source requires to produce an answer.

R$^+$ 21 The system is coupled with one or multiple knowledge sources

R$^+$ 22 The active knowledge sources are loaded at system startup

- **R$^+$ 22.1** Load a list of available knowledge sources from a configuration
- **R$^+$ 22.2** Activate the knowledge sources as defined in the configuration
- **R$^+$ 22.3** Knowledge sources may be activated or deactivated dynamically
- **R$^+$ 22.4** If the client request includes one or multiple parameters specifying knowledge sources that should be employed for the query, consult the knowledge sources specified in the client request

R$^+$ 23 Identify the relevant knowledge sources with respect to the query

- **R$^+$ 23.1** Identify knowledge sources based on the same vocabulary as the request
- **R$^+$ 23.2** Identify knowledge sources retrieving the same type of entities the request searches for

R$^+$ 24 Knowledge can be retrieved from locally stored data as well as from the Web at runtime

- **R$^+$ 24.1** The system can retrieve data from relational databases
- **R$^+$ 24.2** The system can retrieve data from RDF repositories
- **R$^+$ 24.3** The system can retrieve data from SPARQL endpoints
- **R$^+$ 24.4** The system can retrieve data from Application Programming Interfaces (API)s
- **R$^+$ 24.5** Generate SPARQL queries for RDF repositories
- **R$^+$ 24.6** Generate Hypertext Transfer Protocol (HTTP) requests for accessing APIs of external applications
- **R$^-$ 24.7** Generate SQL queries for relational databases

R$^+$ 25 Adapt the generated knowledge base queries to additional parameters included in the client request

- **R$^+$ 25.1** If the client request contains parameters defining the client's location, perform a location-based search for entities in the client's close proximity
- **R$^+$ 25.2** If the client request contains parameters defining the client's location and the query radius, perform a location-based search within the defined radius

R$^+$ 26 If the query's focus is associated with a product and related location-based concepts have been found, search for them as well

R$^+$ 27 If the system does not receive a response from a data source within a given time span, abort the request to the data source

In order to provide additional information about the retrieved results and to present them in an appropriate way, the system performs additional searches for meta data about the found entities. The following data should be provided for entity searches:

R$^+$ 28 Retrieve diverse meta data about result entities
- **R$^+$ 28.1** Retrieve a title for each entity
- **R$^+$ 28.2** Retrieve a URI for each entity
- **R$^-$ 28.3** Retrieve a description (ca. 10-20 words) for each entity if available
- **R$^-$ 28.4** Retrieve a thumbnail for each entity if available
- **R$^+$ 28.5** Retrieve geospatial data for entities if available

R$^+$ 29 In case of fact searches, the system presents the related entity with the requested fact.
- **R$^+$ 29.1** Retrieve a title for the related entity
- **R$^+$ 29.2** Retrieve a title for the requested property
- **R$^+$ 29.3** Retrieve a title for the requested property value
- **R$^-$ 29.4** Retrieve a thumbnail for the related entity if available

3.1.3 Result Processing

The procedure of result processing consists of two phases: First the system needs to merge results from different data sources, then the results are ranked according to an underlying heuristic. A major aspect of the result merging procedure is the identification of redundant results (*instance matching*). We will identify search results by their label as well as geographic results by their location. To rank a set of search results, the system needs a heuristic to compute the relevance of the results towards the natural language query.

R$^+$ 30 Merge the set of search results from different data sources to one result set

R$^+$ 31 Identify redundant search results
- **R$^+$ 31.1** Identify a main label identifying a search result
- **R$^+$ 31.2** Compare the search results' labels with a string comparison algorithm and merge them if their similarity exceeds a certain threshold
- **R$^+$ 31.3** Compare location-based search results on the base of their geospatial data

R$^-$ 32 If a result list's data source provides ranking data, include the ranking information in the ranking process

R$^+$ 33 If geospatial data is available for one or more results and the user provides geospatial data for the query entry point, rank all results based on geospatial data according to their distance to the query entry point

3.1.4 User Interfaces

User interfaces provide the possibility to specify natural language queries and present retrieved search results to the user in a human-readable form. Users may enter a natural language query as well as various meta parameters to specify the request. Furthermore, user interfaces may transmit the client's location in order to proceed location-based searches. This feature may especially be of interest for mobile usage of the system. The way the retrieved data is presented to the user depends on whether the search retrieved a list of general or location-based entities or the fact of a specific entity.

R^+ 34 Users can specify a natural language query as well as further optional meta parameters of the query

- **R^+ 34.1** Users can enter a natural language query in a text form
- **R^- 34.2** Users can define the maximum number of results
- **R^- 34.3** Users can select whether the retrieved search results are displayed in German or English
- **R^- 34.4** The user interface can transmit the client location's latitude
- **R^- 34.5** The user interface can transmit the client location's longitude
- **R^- 34.6** The user interface can specify a list of knowledge sources the system shall consult for the retrieval of a search response

R^+ 35 Present the retrieved search response to the user

- **R^+ 35.1** Entities are presented to the user as a list of results enriched with the meta data title, description and an image
- **R^+ 35.2** If geospatial coordinates have been retrieved for one or more entities, display all location-based entities of the result set on a map which is shown additionally to the result list
- **R^- 35.3** Facts of entities are displayed with the title and a thumbnail of the corresponding entity as well as the retrieved fact name and value
- **R^- 35.4** Present search results to the user in the language specified in the client request
- **R^- 35.5** If no language has been selected in the client request, select a default language defined by the system configuration

3.2 Non-functional Requirements

This section defines the non-functional requirements of the question answering system, which includes the foundations of the system's architecture, its vocabularies and knowledge sources as well as the specification of system interfaces. A major aspect of the system design is the independence towards the underlying data sources. For this purpose, the systemimplements a fully modualrized architecture. A major aspect of the presented approach is the independency with respect to the underlying vocabularies and the knowledge sources for information retrieval. This implies that new lexica can be added or removed dynamically, further vocabularies can be included independently from their underlying data format. For this purpose we propose a modularized architecture that separates components querying external data sources from the main system.

3.2.1 System Design

R$^+$ 36 The system consists of a main system for processing a natural query as well as a set of modules for querying external data sources

R$^+$ 36.1 The main system consists of three components

- a syntactic parser processing the natural language query
- a query performer consulting knowledge sources
- a result processor generating a result set

R$^+$ 36.2 External data sources are accessed by small modules separated from the main system

R$^+$ 36.3 The system components are realized as reusable software modules

R$^+$ 36.4 The system can be accessed by web applications

R$^+$ 36.5 The system can be accessed by mobile applications

R$^+$ 36.6 Interactions with external data sources are proceeded by separated modules

R$^+$ 36.7 The system's data sources can be modified by adding or removing data sources

R$^+$ 37 Two kinds of external data are included by the system:

R$^+$ 37.1 Vocabularies: Terminological and assertional knowledge included in order to parse the syntax and semantics of a natural language query into a logical intermediate representation

R$^+$ 37.2 Knowledge Sources: Assertional knowledge queried in order to retrieve one or multiple search results with considering the client request

R$^+$ 38 All components are implemented in the Java programming language[1]

R$^+$ 39 The system's lexica are implemented as MySQL[2] databases

[1]http://oracle.com/technetwork/java
[2]http://www.mysql.com

3.2.2 Data Sources

In order to process a client request, the system relies on two kinds of data retrieved from external sources: The first is the terminological knowledge required for the syntactic parser to correctly parse the syntax and semantics of a natural language sentence, i.e. the *vocabulary*. The vocabulary determines the possible input of the parser and therefore embodies the *domain* of the system. The second input of the system is the knowledge source consulted to retrieve answers for the input query. This data represents the possible responses of the system, i.e. the *range* of the system.

Vocabulary Sources In order to parse a natual language sentence, a syntactic parser requires knowledge about the meaning of the words contained in the sentence. Our goal is to use the conceptual world model described by ontologies to compute a logical representation of the information included in a natural language sentence.

R$^+$ 40 The system consults one or multiple lexica in order to recognize the meaning of segments within a natural language sentence

- **R$^+$ 40.1** A lexicon represents the knowledge about classes, properties and concrete instances of a specific domain
- **R$^+$ 40.2** A lexicon associates words or word combinations with semantic entities (tokens)
- **R$^+$ 40.3** Semantic tokens are identified with a unique identifier
- **R$^+$ 40.4** The conceptual data contained in a lexicon has been extracted from an ontology
- **R$^+$ 40.5** The data corpus of a lexicon consists of the classes and properties of an ontology as well as a set of instances described by the vocabulary

R$^+$ 41 Lexicons are consulted by small components separated from the main system (*vocabulary modules*)

- **R$^+$ 41.1** The main system is coupled with at least one vocabulary module
- **R$^+$ 41.2** The main system consults vocabulary modules in order to receive data about the meaning of words contained in the client query
- **R$^+$ 41.3** The main system can add or remove vocabulary modules dynamically at runtime

Location-based NLIs often provide users the possibility to search for various points of interest. An additional valuable feature is the retrieval of locations where a specific entity can be found, e.g. local businesses where a specific product can be acquired. For this purpose the system is able to retrieve data about local businesses for product searches.

R$^+$ 42 Provide a lexicon for retrieving location-based related terms

- **R$^+$ 42.1** The lexicon provides a mapping between real-world objects and certain location-based classes indicating where the respective object can be acquired

Knowledge Sources A major aspect of the system is to enable multiple knowledge sources that are consulted for retrieving answers. Additionally, the system shall be able to include data sources independently from their DBMS, i.e. it should be possible to include data from relational databases as well as LOD sources and APIs. The modularized architecture delegates the query generation to the modules querying the data sources. In this way the system can access a data source independently from its database's internal organization.

R$^+$ 43 The system is based on one or more data sources in order to retrieve results for a query

- **R$^+$ 43.1** The process of querying knowledge sources is performed by modules separated from the main system (*data modules*)
- **R$^+$ 43.2** Each data module is coupled with one particular knowledge source
- **R$^+$ 43.3** Each data module is based on one of the registered vocabularies
- **R$^+$ 43.4** The system can add or remove knowledge sources dynamically at runtime
- **R$^+$ 43.5** The main system is coupled with at least one data module

3.2.3 Interfaces

The process of computing a search request includes the interaction of different components which require well-defined interfaces. The main components interact within the system by transferring data structures representing the result of the previous component's computation. The system's internal communication with data structures takes place between the main components Syntactic Parser, Query Performer, Result Processor and the HTTP interface receiving client requests. The communication will be based on the Representational State Transfer (REST) architectural style introduced by (92).

R$^+$ 44 The communication between components of the main systems proceeds by transmitting data structures

- **R$^+$ 44.1** The Syntactic Parser transmits an intermediate representation of the natural language parse to the Query Performer

- **R$^+$ 44.2** The Query Performer transmits a set of retrieved search results to the Result Processor
- **R$^+$ 44.3** The Result Processor transmits a list of search results to the system's main interface

R$^+$ 45 The intermediate representation depicts the graph-based logical representation generated by the Parser

- **R$^+$ 45.1** The IQR expresses information in a subject-predicate-object triple format
- **R$^+$ 45.2** The IQR contains the request type of a client query (*entity, fact*)
- **R$^+$ 45.3** The IQR indicates the type of entities the client query searches for
- **R$^+$ 45.4** In order to represent a valid query, the IQR must contain at least one non-terminal node or branch, i.e. a variable

Other than the components within the main system, the communication between the main system and vocabulary and data modules proceeds by employing well-defined interfaces. In this way, the components accessing external data sources can be modified easily without the need to modify or interrupt the main system. The communication between the main system and user interfaces proceeds by the employment of a well-defined interface as well. This solution enables the integration of the system within multiple user interfaces independently of the client's platform and operating system.

R$^+$ 46 The communication between the main system and its vocabulary and data modules proceeds by using well-defined interfaces

- **R$^+$ 46.1** Vocabulary modules are accessed via RESTful web services
- **R$^+$ 46.2** A request to a vocabulary module is a HTTP request containing a word or a list of words of the client query as parameters
- **R$^+$ 46.3** The response returned by vocabulary modules is an XML encoded string describing a list of semantic tokens associated with the query terms
- **R$^+$ 46.4** Data modules are accessed via RESTful web services
- **R$^+$ 46.5** A request to a data module is a HTTP request containing a list of triples representing the semantic graph generated by the Syntactic Parser as parameters
- **R$^+$ 46.6** The response returned by data modules is an XML encoded string describing a list of search results that have been retrieved as a response
- **R$^+$ 46.7** The system's main interface is available as a RESTful web service
- **R$^+$ 46.8** A request to the system is a HTTP request containing the natural language query as well as various optional meta parameters for specifying the client request
- **R$^+$ 46.9** The response returned by the system is an XML encoded string describing a list of search results that have been retrieved as a response

Chapter 4

The Model

This section presents the syntactic parsing model for generating a semantic representation of a natural language sentence. The parsing process consists of two phases: First, the natural language input is analyzed and separated into logical semantic units (*tokenization*). This process will be based on the structured contextual data provided by ontologies, recognizing textual entities refering to terminological knowledge as well as concrete instances. The second parsing stage is the semantic linkage of the identified entities and the generation of a logical representation of a query to one or more of the system's knowledge bases (*semantic interpretation*). For this purpose, we will present an approach for generating an intermediate representation of a natural language query in terms of a labeled directed graph in 3 stages with the employment of specified patterns.

4.1 Tokenization

The first step of the presented analysis of a natural language sentence is the identification of the words' meaning contained in the client request. For this purpose, the system will structure the natural language query into smaller units referring to semantic entities such as concepts or real-world entities. Procedures for identifying entities within a natural language query as a preceding step of a deeper analysis have been employed by various works (59) (68). The presented approach will combine the identification of semantic entities in a natural language query with a full computation of possible segmentations of the query. Based on the data corpora provided by ontologies, the Tokenizer identifies elements referring to entities of the terminological model such as classes and properties as well as concrete instances. The tokenization of a natural language sentence is thus the first of two phases of the parsing model[1]. Its purpose is to analyze the input string and to identify logical semantic units (*tokens*) included in the natural language query. Within this context, the Tokenizer has to compute a segmentation of the query where ideally each segment represents a semantic token. For identifying the most probable segmentation, the system has to consider that a natural language reference to an entity may consist of multiple words. Furthermore, the system has to resolve disambiguities and select the most probable tokens for a given query. The tokenization process thus comprises of the computation of all possible segmentations of a natural language query, as well as the association of semantic tokens and the computation of the most probable tokenization.

4.1.1 Segmentation

The segmentation of a natural language sentence is a preparation for the token retrieval procedure and consists of the computation of the set of possible word combinations that coherently refer to a logical entity. The model's intuition is as follows: For parsing a natural language input, the Tokenizer identifies semantic logical units contained in the input sentence. A token is identified by a label which is associated with the token and consists of one or more words which we will call a *term identifier*. For example, in a lexicon employing the DBpedia ontology(93) (94), the term *"university"* is likely to refer to the class `dbpedia-owl:University`, which may indicate a request searching for universities in a specified region or other specifications. However, the term *"university of Koblenz"* refers to a concrete instance, which is probable to indicate a request referring to information about the institution or persons or organizations related to the university. As a term identifying a token can consist of one or multiple words, the purpose of the segmentation is to compute all possible combinations of words with respect to the word order.

We define the segmentation of a natural language sentence as follows: We assume a system receives an input string q consisting of a set of words $\{w_0, ..., w_k\}$ separated by white spaces. One or more words representing a token t_i constitute a term identifier ti_{t_i}. All terms of an input string form the *term sequence ts*.

Definition 4 *(Term Sequence)*
A term sequence ts $(t_0, ..., t_n)$ *is the separation of an input string* $q(w_o, ..., w_k)$ *in* n *terms* $(t_0, t_1, ..., t_n)$*. A term* t_i *consists of* m *words* $(w_0, ..., w_m)$*.*

[1]The term *tokenization* is used frequently in various works and often refers only to the task of separating words in an input query (11). For the scope of this thesis, we use the term *tokenization* for the whole process of identifying logical entities in a natural language query. The term *token* refers analogously not just to a set of words but to a logical unit representing a semantic entity.

In this way, the phrase *"hockey team"* could be segmented in the term sequences *(hockey team)* and *(hockey,team)*. As a term can consist of one or multiple words, there are multiple possibilities to separate an input string into a term sequence. The set of all possible term sequences is called the *term sequence set*.

Definition 5 *(Term Sequence Set)*
The term sequence set tss $(ts_0, ..., ts_m)$ *is the set of all possible term sequences* $ts_0, ..., ts_n$ *of an input string* q.

The sentence *"birds of New Zealand"* could be divided per white spaces into 4 terms (*birds,of,new,zealand*). Alternatively, the input string could also represent a single term (birds of new zealand). The purpose of the tokenization stage is to identify the most probable term segmentation of the input string and the logical entities they refer to. The full term sequence set of the input string *"birds of new zealand"* is depicted in Figure 4.1.

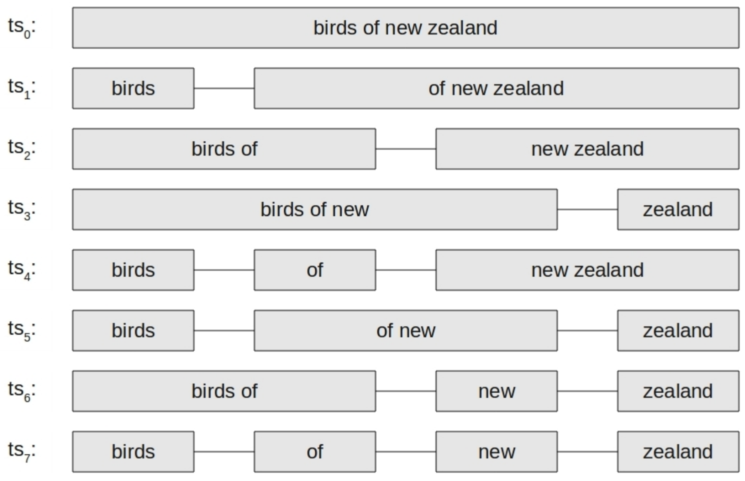

ts0: birds of new zealand
ts1: birds | of new zealand
ts2: birds of | new zealand
ts3: birds of new | zealand
ts4: birds | of | new zealand
ts5: birds | of new | zealand
ts6: birds of | new | zealand
ts7: birds | of | new | zealand

Figure 4.1: Term Sequence Set of the input query "birds of new zealand"

4.1.2 Token Retrieval

After all possible term combinations have been computed, the parser computes semantic tokens from the term sequences. That is, the parser searches for logical units corresponding to the terms of each term sequence and generates a sequence of semantic tokens. To associate words with semantic entities, the Tokenizer consults one or more lexica that map natural language terms to semantic units with a certain probability. The identification of semantic entities based on the data of ontologies has been proposed by (68). We will apply the concept of associating ontological elements with natural language terms to the concept of computing a set of possible sentence segmentations and expand the associations by mapping probabilities. Additionally, we will enable the identification of vocabulary-independent query constraints. For modeling the information contained in a natural language query, we will employ the data model provided by ontologies, which describes knowledge in terms of terminological knowledge (TBox) as well as assertional knowledge providing facts about those instances (95). Tokens can thus be terminological entities described by an ontology such as classes (e.g. *"City"*) or properties (e.g. *"population"*) as well as conrete instances (e.g. *"Koblenz"*) of a class. Furthermore, segments of a NL query may refer to vocabulary-independent elements indicating query constraints such as numeric delimiters, date intervals or currencies (e.g. *"with more than 3 ..", "in january", "tomorrow"*). Query constraints will be identified as an own tokens which we will call *base tokens*. Formally, we define a semantic token as follows: A token $t(ti, \pi, u, r)$ is a quadruple consisting of a term identifier ti_t, a token type π, a unique identifier u and a relevance r. A token represents a mapping of the form

$$ti_t \rightarrow (\pi, u, r)$$

between a term identifier ti_t and a resource identified by a URI u, the token type π and the relevance r. The URI u indicates a unique identification of the resource on the Web. The token type π $[c, r, i, b]$ represents the semantic role of the token and can either be class(c), role(r), instance(i) or base(b). The term identifier ti_{t_i} depicts the natural language identifier of the token. The relevance r is a decimal value of the interval $[0; 1]$ indicating the probability that the given mapping is correct. The token type of a token indicates whether a token is part of the terminological vocabulary (classes and roles), an instance or a specific value(base). Base tokens t_b are tokens indicating a query constraint, e.g. a specific quantity, minimum or maximum values of property or a time interval. In case of base tokens, the URI additionally contains a value indicating the query constraint of the token, e.g. *">3","january"*. For retrieving tokens associated with the given term sequences, a lexicon is consulted that contains mappings between terms and semantic tokens. The lexicon contains term mappings about all known resources of the knowledge base of the form

$$ti \rightarrow T(\pi, u, r)* \tag{4.1}$$

where a term identifier ti is associated with one or multiple tokens. The mappings contained in the database represent references to entities of an ontology and have been extracted in advance. The Tokenizer retrieves the most probable token T_{max} of a term t_i by selecting the mapping

$$t(t_i)_{max} = \arg\max r(t_i, T) \tag{4.2}$$

, i.e. the mapping that maximizes the relevance r of a given term t_i and a token T.

We assume a lexicon contains term mappings refering to the class *Bird* and the instance *New Zealand* of the form "bird"→(c,dbpedia-owl:Bird,0.99), "new zealand"→(i,dbpedia:New_Zealand,0.99). The term sequence set computed before produces the token sequences shown in Figure 4.2. Generated tokens are depicted with their token type (here: class(cls) or instance(inst)), their URI and their relevance values.

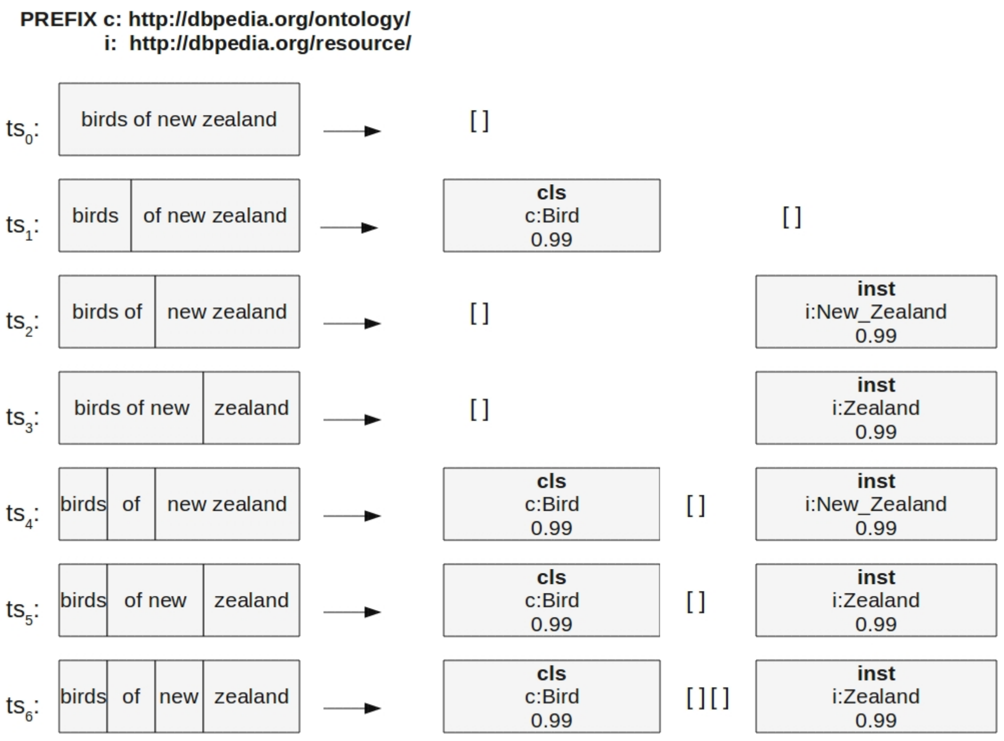

PREFIX c: http://dbpedia.org/ontology/
i: http://dbpedia.org/resource/
ts0: birds of new zealand → []
ts1: birds | of new zealand → cls c:Bird 0.99 []
ts2: birds of | new zealand → [] inst i:New_Zealand 0.99
ts3: birds of new | zealand → [] inst i:Zealand 0.99
ts4: birds | of | new zealand → cls c:Bird 0.99 [] inst i:New_Zealand 0.99
ts5: birds | of new | zealand → cls c:Bird 0.99 [] inst i:Zealand 0.99
ts6: birds | of | new | zealand → cls c:Bird 0.99 [][] inst i:Zealand 0.99

Figure 4.2: Token Sequence Set generated from the term sequences of Figure 4.1

Tokenization Computation After generating tokens for all available term sequences, the Tokenizer selects the token sequence which represents the most probable parse for the input query. For this purpose, the probability of each token sequence which has at least one token is computed. We will call a token sequence associated with a probability a *tokenization*.

Definition 6 *(Tokenization)*
A tokenization T of a term sequence ts is a token sequence $(t0, ..., tn)$ with a probability p. The most probable tokenization T_{max} for ts is the most probable tokenization of a term sequence set tss.

After computing all possible tokenizations $T_0, ..., T_n$ of the term sequence set $ts_0, ..., ts_n$, the Tokenizer selects the most probable tokenization $T(q)_{max}$ as the tokenization of the input string q. The most probable tokenization is selected according to the follow criteria:

1. Select the tokenization which associates the highest fraction of the input string to semantic tokens
2. Select the tokenization with the least tokens

Intuitively, the parser computes the tokenization that is able to interpret the whole NL query with as few tokens as possible. For example, the tokenizations computed for the term sequences *(hockey team)* as well as *(hockey, team)* would both result in valid tokenizations. If the same probability has been computed for both tokenizations, the Tokenizer selects the one which identifies the sentence with fewer tokens. Since the tokenization for the first term sequence realizes an interpretation of the full phrase with only one token, the first tokenization would be preferred. The named criteria can be realized in a formula that 1) computes the sum of the weighted retrieved tokens and 2) in case of equal probabilities selects the tokenization with the fewer tokens, i.e. choses the tokenization providing the simpler solution for retrieving the same result. We assume a token is an interpretation of a term, which consists of a set of words. Let t_i be a token, TI_{t_i} the set of words its term identifier contains and W be the set of words included in the input sentence. We say the weight w_{t_i} of a token t_i is

$$w_{t_i} = \frac{|TI|}{|W|} \tag{4.3}$$

the number of words the term identifier contains divided by the number of words of the whole input sentence, i.e. the fraction of the input sentence the token represents. In this way, parses that take into account all words of the input string receive higher weights. In case of the sample sentence *"birds of new zealand"*, the weight of the semantic token identified for the term "birds" would thus be computed as follows: $w_{t_1} = \frac{TI_{t1}}{W_q} = \frac{1}{4} = 0.25$. The probability p_{t_i} of a tokenization t_i is computed as the sum of the weighted token probabilities:

$$p_{t_i} = \sum_{i=0}^{l} w_{t_i} p_{t_i} \tag{4.4}$$

Finally, the most probable tokenization $T_{max}(q)$ of a natural language sequence q is the one which maximizes the probability $p(T,q)$

$$p(T,q)_{max} = \arg\max\ p(T,q) \tag{4.5}$$

If the maximal probability $p(t,q)_{max}$ is reached by multiple tokenizations, the association featuring the fewest tokens is selected. The result of the token retrieval procedure $T_{max}(q)$ is the most probable tokenization t_{max}.

The probabilities of the token sequences generated before are calculated as follows:
$T_0 : p_{T_0} = 1.0 \cdot 0.00 = 0$
$T_1 : p_{T_1} = 0.25 \cdot 0.99 + 0.75 \cdot 0.00 = 0.2475$
$T_2 : p_{T_2} = 0.5 \cdot 0.00 + 0.5 \cdot 0.99 = 0.495$
$T_3 : p_{T_3} = 0.75 \cdot 0.00 + 0.25 \cdot 0.99 = 0.2475$
$T_4 : p_{T_4} = 0.25 \cdot 0.99 + 0.25 \cdot 0.00 + 0.5 \cdot 0.99 = 0.7425$
$T_5 : p_{T_5} = 0.25 \cdot 0.99 + 0.5 \cdot 0.00 + 0.25 \cdot 0.99 = 0.495$
$T_6 : p_{T_6} = 0.25 \cdot 0.99 + 0.25 \cdot 0.00 + 0.25 \cdot 0.00 + 0.25 \cdot 0.99 = 0.495$

The highest probability is reached by $p_{T_4}(0.7425)$, the tokenization $T_{max}(q)$ is therefore T_4.

4.2 Semantic Interpretation

The major procedure of the parsing model is the transformation of the retrieved tokens into a semantic representation of the query. In this section we will present a 3-staged approach to generate a graph-based representation of a natural language query using specified patterns. The presented approach will apply the concept of graph-matching employed by the SPARQL query language (69) to the interpretation of a natural language sentence in a novel way. The base for storing natural language information in a logical form is a graph-based representation fulfilling the requirements defined in section 3.2, which we will refer to as a *semantic graph*.

An undirected graph as defined in (14) is a set of points (nodes) with lines (edges) connecting some of the points. Applied to natural language processing, we define nodes to represent entities of terminological or assertional knowledge (such as *classes*, *properties*, *instances* or *query constraints*), while the edges represent verbs or prepositions indicating relationships between nodes (*roles*). A semantic graph G is a directed labeled graph consisting of a set of nodes $n_0, .., n_m$ connected with a set of branches $b_0, ..., b_k$. Each node is connected with at least one branch to the graph. The intuition of a graph-based representation of a natural language sentence is the representation of information in terms of nodes connected by labeled, possibly weighted edges: For example, the query *"movies directed by Alice"* could be modeled as a set of entities represented by a non-terminal node (*?x*) connected with a class node *movie* and an instance node *Alice.* The edge labels represent the relations between the nodes, in this case *typeOf* and *directedBy.* The analogous triple-based representation employing a sample ontology can be formulated as follows:

?x rdf:type sample:ontology/Movie .
?x sample:property/directedBy sample:resource/Alice

The semantic interpretation model developed in this thesis is a finite-state-based mechanism that transforms a token sequence into a semantic graph in 3 stages. After receiving a token sequence as described in section 4.1, the Semantic Interpreter generates a graph presenting relations between entities as connected nodes of a graph. The semantic interpretation proceeds as follows: The interpretation begins by identifying semantic entities included in the NL query. Then, relations between the computed entities are identified in terms of an information request. Finally, the query's main information request (*focus*) is identified. For the sample query mentioned above, the semantic interpretation first identifies the semantic entities *Alice* and *Movie* contained in the query. To model the query in terms of a graph representing an information request, the system interlinks the identified entities by creating an element representing a list of unknown entities which are of type *Movie* and are directed by *Alice.* As the lexicon identifies the term *directed by* in the query as a verb, the system associates *directed by* with the entity *Alice*, resulting in a directed edge interlinked with the unknown entities *?x.* Finally, these entities are marked as the graph focus to determine what the graph searches for. Applied to the graph-based intermediate representation, the steps performed by the Semantic Interpreter can be described as the stepwise generation of the elements of a graph:

- Generation of the graph **nodes**
- Generation of the graph's **relations**
- Identification of the graph's **focus**

Figure 4.3 shows the 3-staged concept with the stepwise generation of the graph structures nodes, relations and the graph's focus. The major technique to generate each stage of the semantic graph is the employment of specified patterns. Patterns represent transformation rules similar to formal grammars as introduced in section 2.1, that are specified for receiving a sequence of tokens and generating a graph-based semantic interpretation. Formally, we define a pattern as a mapping p of the form

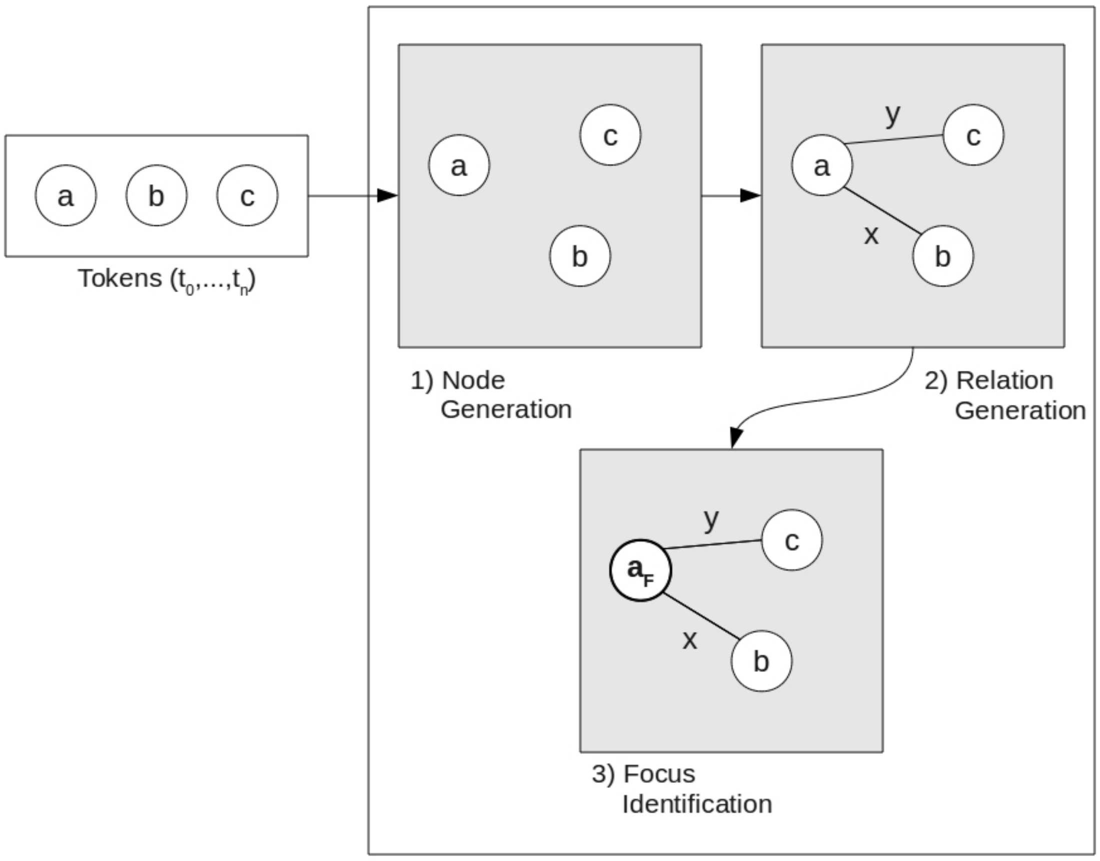


Figure 4.3: 3-staged graph generation from a token sequence

$$p: \ E- > A \quad (\sigma) \tag{4.6}$$

where E is an input sequence of elements called the *expression*, A is the output token sequence indicating the expression's transformation called the *annotation*. σ is a decimal value indicating the probability the given pattern is correct, called the pattern's *truth value*.

The expression e $(t_0, ..., t_n)$ is a sequence of one or multiple entities $(t_0, ..., t_n)$ identified with an entity type π and an identifier u. The annotation a $(t_0, ..., t_m)$ contains a sequence of elements $(t_0, ..., t_m)$ identified with an identifier u. The annotation may contain elements occurring in the expression or add new elements, indicating the transformation of the expression's tokens into another stage. Tokens transformed from the expression are denoted with the same identifier u as the corresponding identifier of the expression. The annotation contains at least one element from the expression. The truth value θ $(0;1)$ indicates the probability that the given correlation is correct. The presented concept of patterns is similar to the concept of probabilistic context-free grammars introduced by (13) (1) (14) (12) in section 2.1 in terms of representing transition rules. The patterns presented in this thesis are specified for transitions between the semantic graph generated by the model in 3 stages.

The expression of a pattern embodies a token structure which is compared to an input token sequence, whereupon the annotation defines the expression tokens' transformation. Intuitively, a pattern's expression could be described as the structure for identifying a specific part of an input token structure, the annotation as the description of how to parse it into a semantic graph.

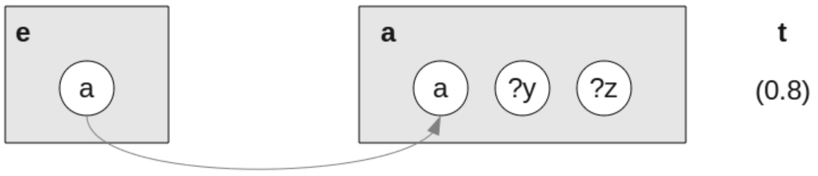


Figure 4.4: Simple pattern indicating the generation of a graph branch from a token

Figure 4.4 shows a simple pattern transforming a token into a branch including two unspecified tokens. A token identifier (a) is employed to map elements from the expression to the annotation. The output of the pattern employment is a branch of the form (a - ?y - ?z), where the first element has been mapped into a new structure and two new elements have been added. In order to transfer an input sequence into the next stage, the parser identifies the most probable pattern and maps the input sequence's elements to entities of the pattern's expression. The most probable pattern $p(ts)_{max}$ for a token sequence ts is the one whose expression resembles the token sequence the most, that is, the pattern p_i which maximizes the similarity function $p(ts, e_{p_i})$ of a token input sequence t and the pattern's expression e_{p_i}

$$p(t,e)_{max} = \arg\max p_i(t,e) \tag{4.7}$$

The similarity between a token sequence and an expression is computed in terms of a string similarity function. Adapted to the concept of comparing token sequences, we define to compare tokens by their token types π_i. A token is considered different from another token if its token type differs from the other one. We define that the token comparison is the token similarity γ as follows:

$$\gamma(t_0, t_1) = \begin{cases} 1 & \text{if } \pi_0 = \pi_1 \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

After the most probable pattern has been computed, the system generates a mapping between the token sequence and the expression sequence

$$m := ts \rightarrow e \tag{4.9}$$

which assigns input tokens to expression tokens. The output sequence is indicated by the pattern's annotation by generating an output sequence from the expression. If the annotation contains an element of the expression for which no element of the input sequence could be found, the parser generates a non-terminal node. Each phase of the parsing model employs patterns with specialized structures. Analogously to the phases of the parsing model, we distinguish node patterns, focus patterns and relation patterns.

4.2.1 Node Generation

The first step in the graph generation process is the identification of nodes. Nodes are the base elements of a graph and represent one or multiple entities. A node can be a class (e.g. *"University"*), an instance (e.g. *"Koblenz"*) or a value constraint such as a specific amount or a time interval (e.g. *"2","50-100","june","2010-2012"*). Additionally to these terminal nodes we include nodes of an undefined kind representing one or multiple unspecified entities. These non-terminal nodes represent variables in the query, i.e. a set of entities which are described by the relations to other nodes within the semantic graph. Intuitively, non-terminal nodes represent a set of unknown entities in the query indicating what the natural language query of the client searches for. Formally, we describe graph nodes as follows: A graph node n_i (θ, π, u, p) is a quadruple representing one or multiple entities with the node indicator θ, the identifier u, the node type π and the position p. The boolean variable θ indicates whether n_i represents a terminal or a non-terminal node. A terminal node represents a single resource, the value for the identifier u is the resource's URI. The node type π_n represents the entity's semantic role $(class, role, instance, constraint)$. Non-terminal nodes represent a set of entities and are called variables. They receive a unique alphanumeric value for u, their node type π is undefined. The position p indicates the node's original position in the natural language query.

Analogously to the token types presented in section 4.1, graph nodes are equipped with node types indicating whether the resource is a class(c), a property or role(p), an instance(i) or a base node(b). To generate a set of graph nodes of the token sequence produced in the tokenization stage, the parser employs a set of patterns transforming tokens into graph structures called *Node Patterns.*

Definition 7 *(Node Pattern)*
A node pattern p_n is a pattern indicating how a token is transformed into a graph node. A node pattern consists of an expression $e_n(t)$ consisting of a single token t, an annotation $a(n[, n_p, n_i])$ indicating at least one node that is generated from the expression token, , and a truth value θ. The annotation may also generate a triple statement, i.e. a branch, from a single token.

The node type and the URI are analogous to the token type and the URI of semantic tokens of the Tokenization. Node Patterns constitute an intermediate stage in the graph generation process, representing nodes or simple branches which can be derived from single semantic tokens. In the simplest case, a node pattern transforms a semantic token into a single node. This may be the case for instances (e.g. *"Koblenz"*, *"Alice"*), which are represented by instance nodes. Analogously, a role token can be interpreted as a labeled branch; for example, a phrase *"population of"* can be interpreted as a labeled branch connecting two non-terminal nodes (*?x - p(populationOf) - ?y*). Followingly, the query *"population of Koblenz"* would result in the generation of an instance node *i(Koblenz)* and a labeled branch *p(populationOf)* in the first stage of the semantic interpretation procedure. In case of class tokens, the system may already proceed a more advanced interpretation: As a class token such as *university* may indicate a query searching for entities of the type `university`, a node pattern could define a non-terminal node *?x* connected with a class node *university* with a labeled edge *typeOf.* A branch is also defined in case of base tokens: A query *"Name all cities with more than 100.000 inhabitants"* containing a restriction of the population size could be transformed into a labeled branch p(*populationSize*) and a base token constituting a numeric restriction b(> *100.000*). The following listing shows a set of possible node patterns:

$$p_0 := \quad c[a] \ \rightarrow \ ?x \ \ p(rdf:type) \ \ c[a] \qquad (0.95)$$
$$p_1 := \quad i[b] \ \rightarrow \ i[b] \qquad (0.9)$$
$$p_2 := \quad p[c] \ \rightarrow \ ?x \ \ p[c] \ \ ?y \qquad (0.85)$$
$$p_3 := \quad b\{v\}[d] \ \rightarrow \ ?x \ \ p[d] \ \ b\{v\}[d] \qquad (0.95)$$

p_0 indicates the interpretation of a class token, which corresponds to a non-terminal node ($?x$) of the corresponding class (c). The identifier indicating how the token is included in the pattern annotation is depicted in brackets ([a]). A class token *university* would thus be transformed into a labeled branch of the form (*?x - rdf:type - university*). p_1 produces a direct transition from an instance token into an instance node (i), while p_2 generates a labeled branch p with undefined nodes from a role token. p_3 describes the processing of a base token, which corresponds to the generation of a branch with the token's label as the branch's label and its value as the branch's object node. A phrase *".. in january"* would thus be interpreted as a branch of the form (*?x - date - january*). The employment of node patterns on a token sequence is depicted in Figure 4.5, while the curved arrows imply the association of tokens with pattern elements. The transformation that computes the annotation (a) from the expression (e) results in the first graph stage G_0.

The query "universities in Koblenz" has produced the tokens $(t_0(c), t_1(i))$ representing a class token (*c:University*) and an instance token (*i:Koblenz*) employing a lexicon based on the DBpedia ontology. If we employ the node patterns listed above on the generated tokens, the parser associates the node patterns p_0 and p_1 with the corresponding tokens, thus creates a mapping of the form

$$c : University \rightarrow a(p_0)$$

$$i : Koblenz \rightarrow b(p_1)$$

The employment of the node patterns p_0 and p_1 on the token sequence is shown in Figure 4.5. The generated graph contains a branch indicating a non-terminal of the requested class ("university") as well as a single instance node ("Koblenz"):

PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
c: http://dbpedia.org/ontology/
i: http://dbpedia.org/resource/

$$b : \{?x \ - \ rdf : type \ - \ c : University\}$$

$$n : \{i : Koblenz\}$$

4.2.2 Relation Generation

The second stage of the semantic interpretation approach is the generation of the graph's edges, i.e. the identification of the nodes' relations. The purpose of the *relation generation* is the identification of connections between the available nodes, which we will call the *graph branches*. A branch is characterized as a labeled edge between two nodes, with its label representing the predicate expressing the nodes' relation. Analogously to graph nodes, branches can represent a specific property (*terminal branch*) or an undefined relation between two nodes (*non-terminal branch*). We will define a graph branch $b(n_0, n_1, \theta, p)$ of a graph g as a quadruple connecting two graph nodes n_0, n_1 with a property p. The boolean value θ

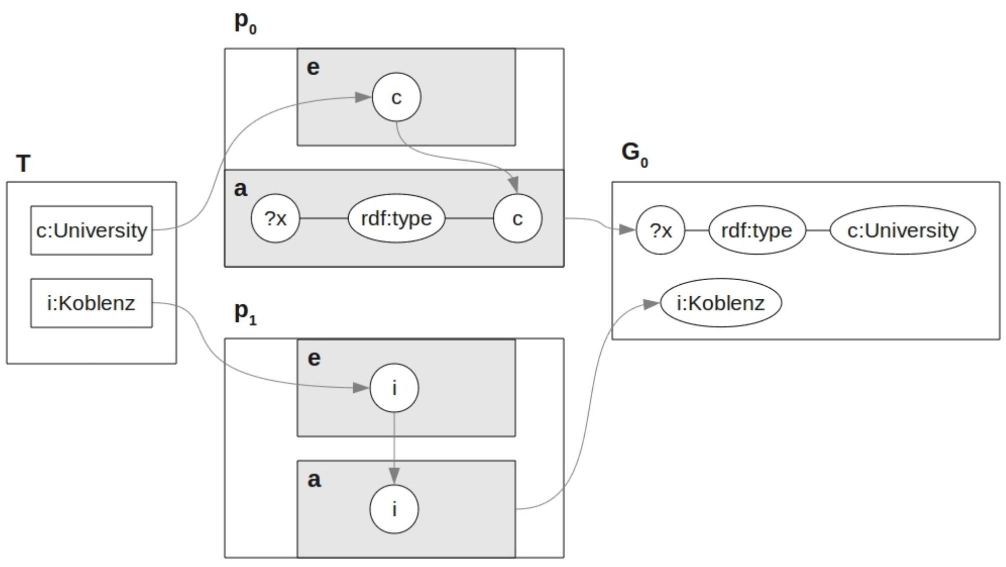


Figure 4.5: Node Pattern employment on a token sequence (c,i)

indicates whether the property specifies a specific or an arbitrary property. Analogously to graph nodes, we distinguish between terminal branches representing a specific property where p is the property's URI, and non-terminal branches representing an undefined relationship, where p is identified with a unique alphanumeric id. The base for identifying relations between nodes are *relation patterns*, which indicate how graph nodes can be connected by branches. Other than node patterns, relation patterns receive multiple elements as their input parameters and map them to a pattern's expression.

Definition 8 *(Relation Pattern)*
A relation pattern $p(e, a, v)$ consists of an expression e, an annotation a and a truth value v. The expression $e(n*)$ is a node sequence consisting of at least one node. Each node is identified by a unique id and a node type. The annotation $a((n_s, n_p, n_o)*)$ is a set of branches indicating the transformation of the expression into one or more branches. The annotation must contain at least one node from the expression. The truth value $v(0; 1)$ indicates the probability that the given correlation is correct.

The annotation indicates how a set of nodes generated in the first semantic interpretation stage is transformed into a set of branches. As each token in the expression is identified with an ID, nodes of the expression are identified with the same ID in the annotation. While the node generation stage identified single tokens, the relation generation stage identifies cohesive node structures which require a precise procedure to map the elements into the next phase. For this purpose, the parser generates a mapping between graph nodes and elements of a pattern's expression. If an element of the pattern's expression could not be mapped to a graph node and is transformed within the annotation, the parser generates a non-terminal element. The similarity computation between the nodes of the first graph stage G_0 and the pattern expression proceeds by computing the number of matching nodes. By employing the relation pattern whose expression is most similar to the graph nodes, the Semantic Interpreter generates the semantic relations that are most likely to represent the information request included in the client request.

A simple relation pattern indicating the transformation of a non-terminal node, a role token and an instance token *(?x,p,i)* into a cohesive branch with a truth value of 0.95 could be realized as follows:

$$p_r := (\ ?(x)\ .\ i[b]\ ) \rightarrow (\ x\ -\ ?p\ -\ b\ )\ ,\ (0.95)$$

The pattern is employed on the graph $([?x - rdf : type - c : University]\ ,\ [i : Koblenz])$ as follows: The parser maps the non-terminal node $(?x)$ and the instance node $(i : Koblenz)$ tokens to elements of the pattern's expression:

$$(\ ?x \rightarrow x\ ,\ i : Koblenz \rightarrow b\ )$$

The generated branch indicates an unspecified relation between the non-terminal node and the instance node of the form *(?x ?y i)*. The mapping process from the graph G to the pattern's expression e and the employment within the annotation a is depicted in Figure 4.6.

4.2.3 Focus Identification

After generating the graph's nodes and edges, the focus identification marks the crucial step in realizing the representation of a search request within a graph. Other than the representation of a general set of relations within a graph, the realization of a search request requires an element identified as the query's major element indicating the requested element, called the graph's *focus*. The focus determines the major request entities or facts that are described by the graph, that is, the elements the query searches for. We define the focus f of a semantic graph G as an entity of the graph representing the elements that are to be retrieved. A focus can either be a node n_f or a branch b_f. Each graph can have only one focus. Though the various possibilities what a natural language query can search for, we can distinguish from a graph-based view two base forms of query types: A natural language query can search for *resources* (persons, cities, etc.), i.e. *nodes*, or the

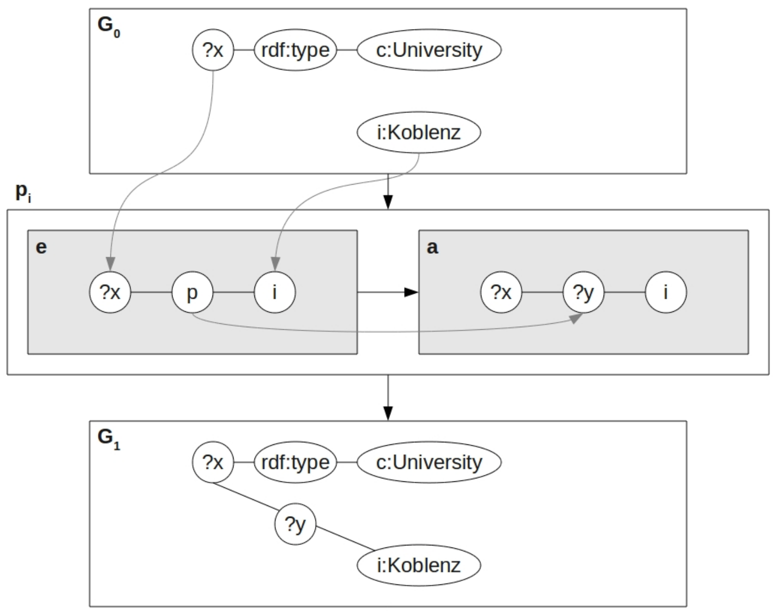


Figure 4.6: Relation Pattern Employment

query searches for a *relation* of an entity, such as the value of a specific property or the relationship between two nodes (birthday of a person, a city's population size, etc.), i.e. a *branch*. Thus we will distinguish between two types of information our system can search for, called the *request type* r_q of a query q:

- **Entities**: A set of resources described by the properties of the NL query (e.g. *"universities in Koblenz"*)
- **Facts**: Properties of a specific resource (e.g. *"weight of an apple"*, *"population of Munich"*)

Whether a query searches for entities or facts, is determined by the focus element, which may either be a node or a branch. A focus node represents the entities that are to be retrieved. The semantic graph represents an *entity request*. A relation between two nodes identified as the focus represents a query for a certain property of a node. The semantic graph represents a *fact request*. For example, a query searching for *universities* would result in a semantic graph comprising of a non-terminal node (*?x*) connected by a branch (*typeOf*) and a class node (*University*). The information request of the graph is depicted by the non-terminal node (*?x*), the query represents an entity request. However, a query *"age of Bob"* would result in an instance *Bob* connected with a branch *age*. Identifying the branch *age* as the graph's focus, the query represent a fact request. The focus of a token sequence is identified with *focus patterns*, which identify a graph's focus as well as its request type.

Definition 9 *(Focus Pattern)*
A focus pattern p_f is a pattern identifying the focus of a graph G. It consists of an expression e_f consisting of a token sequence ts, an annotation $a_{f,r}$ (f), indicating an element of the expression as the graph's focus and a truth value θ.

An additional feature is added with the concept of *focus types*: If the focus node is connected with a class node specifying the type of the focus node, the class node's URI is called the focus type of the parse. The focus type of a graph identifies the type of the requested entities, i.e. what the query searches for (e.g. persons, stores). In this way, the syntactic parser's succeeding components will be able to compute whether a knowledge source of the system is likely to contain an answer to the query. By consulting only knowledge sources that are considered relevant for the client query, the system's efficiency will be increased significantly.

Let p_f be a focus pattern of the form

$$p_f := \ ?x[a] \ \ rdf : type[b] \ \ c[c] \ \rightarrow \ f = a$$

identifying a non-terminal node (*?x*) specified as a class as the graph's focus. The parser employs the focus pattern on the graph computed above by generating the following node mapping:

$$(\ ?x \rightarrow a \ , \ rdf : type \rightarrow b \ , \ c : University \rightarrow c \ )$$

The expression maps token $?x$ to the annotation's focus, identifying the non-terminal $?x[a]$ as the graph's focus and *c:University* as the graph's focus type:

$$f = ?x$$

The focus pattern employment is shown in Figure 4.7. After generating the semantic graph, the syntactic parser transforms the graph into an intermediate query representation that fullfills the requirements specified in chapter 3. Analogously to the RDF triple format, the IQR represents query arguments in the subject-predicate-object format. Additionally, it contains parameters specifiying whether the request searches for entities or facts (request type (r)) and the focus type (f) of the query. The graph generated before is transformed into an IQR as follows:

```
q:= {
        r: e,
        f: c:University,
        {
          ?x   rdf:type   c:University .
          ?x   ?y          i:Koblenz
        }
    }
```

Classification of the approach The presented approach for parsing a natural language sentence into a graph-based intermediate representation has combined techniques of shallow and deep parsing in a novel way. Other than the transition rules expressed by formal grammars introduced in section 2.1, graph patterns are specified for transforming intermediate graph stages in a full semantic graph identified by a set of nodes, branches and a graph focus. Additionally, the presented approach is able to identify the request type as well as the type of entities the query searches for.

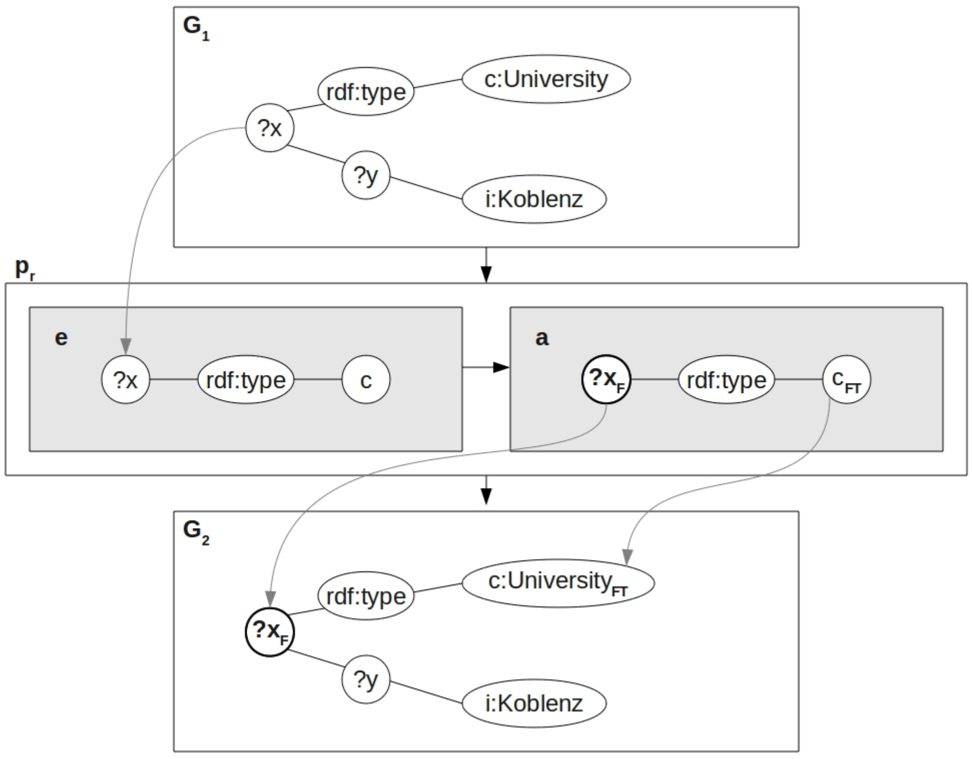


Figure 4.7: Focus Pattern Employment

Chapter 5

Architecture

This chapter presents the system's architecture, consisting of the main components, a set of modularized components for accessing external data sources as well as the interface specifications for the system components' communication. The system consists of three main components: The syntactic parser generating a logical representation of the natural language query, the query performer consulting the system's knowledge sources, and the result processor for computing a request's result set. To realize a system with flexible vocabulary and knowledge sources, we propose a modularized system architecture that proceeds interactions with external data sources by customized modules. An overview of the architecture's components is depicted in Figure 5.1. A client query is processed as follows: As soon as a request comes in, the input is parsed and transformed into a logical semantic representation. For this purpose the syntactic parser employs several lexica storing data about the meaning of terms. The generated logical representation of the query is transferred to the Query Performer, which identifies and consults the appropriate knowledge sources with respect to the query's focus in order to retrieve results for the query. Finally the different result sets from the knowledge sources are merged and ranked by the Result Processor. Section 5.2 will describe the system's interfaces, including the internal communication with data structures and the procedural communication with separated modules, as well as the communication with external applications.

5.1 Components

The system's main system consists of the components Syntactic Parser, Query Performer and Result Processor. These components of the main system reflect the three major tasks proceeded by the distributed QA system: 1) The processing of the natural language query and the generation of an intermediate logical rep-

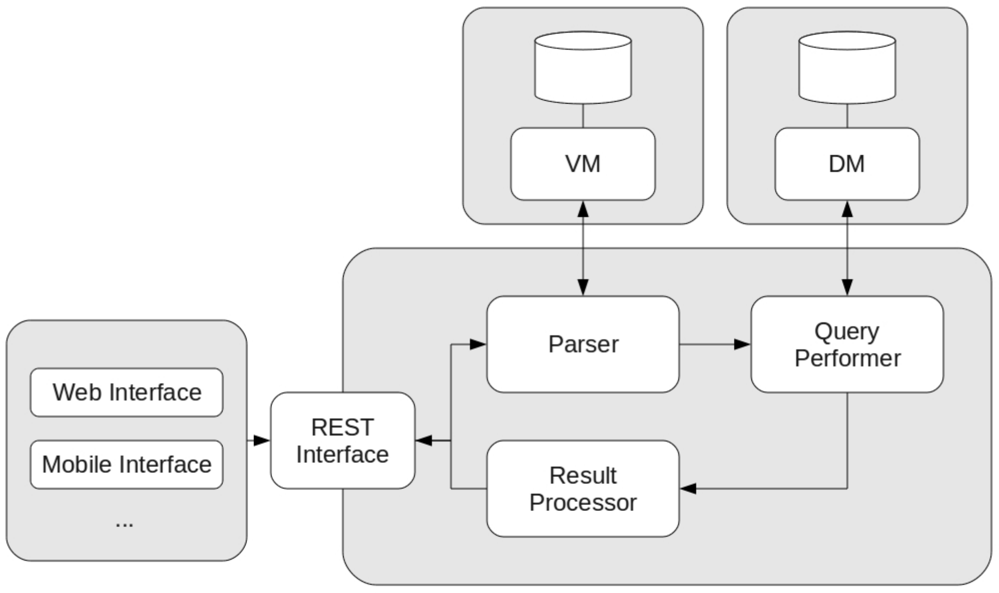


Figure 5.1: System Architecture consisting of the three main components Syntactic Parser, Query Performer and Result Processor

resentation, 2) the retrieval of knowledge from different data sources, and 3) the mergence of the retrieved responses and the computation of a heuristic-based response ranking. The main components thus correspond to the main components of a question answering system proposed by (10) (4) in terms of the major processes proceeded by a NLI. As defined in section 3.2, a major goal of the system design is the system's independency towards its vocabularies and knowledge sources. This goal is achieved by separating processes consulting external data sources from the main system and proceeding them by small, flexible modules. Two of the system's components bring to bear the concept of separated components for querying external data sources: The syntactic parser consults modules for retrieving tokens from natural language (*vocabulary modules*), the query performer retrieves search results from modules querying the system's knowledge sources (*data modules*).

5.1.1 Syntactic Parser

The syntactic parser of the system performs the semantic processing of the natural language and implements the model presented in chapter 4. It receives a natural language query as well as diverse meta parameters from the user interface, parses the query and generates an intermediate query representation of the query. The parser consists of two modules, one for generating a token sequence from the natural language input (Tokenizer) and a module generating a semantic intermediate representation of the query in the form of a labeled graph (Semantic Interpreter). The parser architecture is depicted in Figure 5.2: The main component comprises of the Tokenizer and Semantic Interpreter. From the most probable tokenizations $T*$ computed by the Tokenizer, the Semantic Interpreter generates a set of semantic graphs and transmits it to the Query Performer in an appropriate data structure, the intermediate query representations $IQR*$.

Tokenization The natural language segmentation and token retrieval procedure are computed as described in chapter 4. For retrieving tokens from the term sequences, the Tokenizer needs to consult lexica associating terms with semantic entities. As defined in section 3.3, the goal of a domain-independent NLI requires the possibility to include one or multiple vocabularies in the system. To identify semantic tokens within a natural language query, the Tokenizer is coupled with one or multiple components accessing lexica of different vocabularies which we will refer to as Vocabulary Module (VM)s. The component receives a natural language query q, whereupon the Tokenizer computes the term sequence set and transmits them to its registered vocabulary modules. The Tokenizer generates a tokenization for each vocabulary, i.e. transmits the term sequence set to each vocabulary module separately. VMs are specialized to access their corresponding lexicon and are accessed via HTTP requests using a well-defined interface, thus VMs can contribute to the token retrieval process independently from their underlying database management system.

Vocabulary Modules The separation of the process of consulting external data sources enables the system to employ multiple vocabularies which can be added or removed dynamically. In order to manage the system's vocabularies, the Tokenizer consults a configuration listing the system's available vocabulary modules and specifying which VMs are activated. VMs are employed by the parser for retrieving tokens associated with the term sequences computed in the sentence segmentation process. A VM receives a term sequence from the Tokenizer, whereupon the most probable tokenization is computed as described in section 4.1. Each VM consults a lexicon, i.e. a database associating terms with resources on the Web. To accelerate

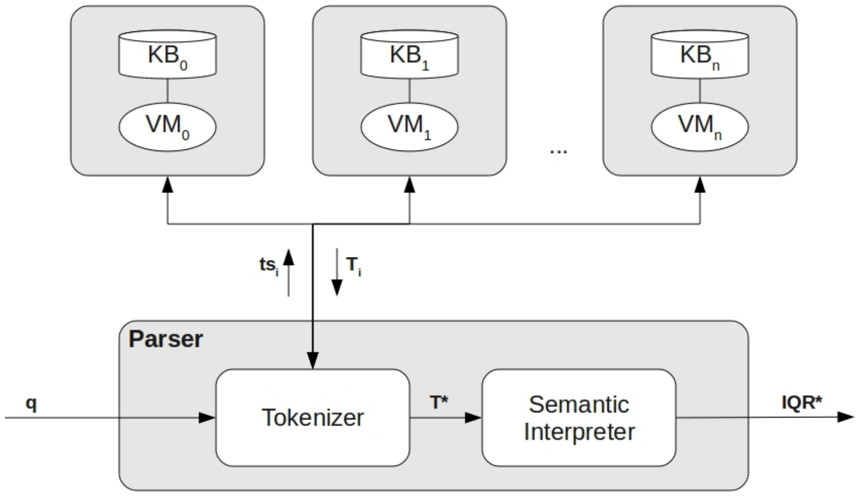


Figure 5.2: Architecture of the Syntactic Parser with modularized access to system vocabularies

the query processing speed, the vocabulary's term mappings have been extracted in advance by the VM and can be queried efficiently.

A resource is identified by its token type such as class, property or instance. Additionally, we will add the URI of the resource to clearly identify the token associated with the term. Thus a database tuple of the lexicon consists of a term, a token type (class, property, instance, base), a URI and a relevance indicating the correlation between a term and a semantic token. The construction of a VM's data corpus of term mappings requires the identification of terms associated with entities contained in the employed ontologies, such as the extraction of labels of classes and properties.The vocabulary database is queried and updated by its corresponding vocabulary module and is completely independent from the main parsing system. In this way the organization of the database is flexible and can easily be modified by updating the corresponding vocabulary module without the need to interrupt the main system.

Semantic Interpretation For generating a graph representation of the natural language query, the parser employs graph patterns in a 3-staged, transition-based parsing model as described in section 4.2. The Semantic Interpreter receives a set of tokenizations $T*$ from the Tokenizer, where each tokenization represents the interpretation of the NL query on the base of its corresponding vocabulary. The semantic interpreter generates a request for each received tokenization. In this way, the system is able to be coupled with multiple vocabularies, which significantly increases the scope of possible client queries. The result of the syntactic parsing process is a set of duples (v_i, r_i) consisting of a request r_i for each vocabulary v_i.

5.1.2 Query Performer

The information retrieval procedure of the system takes place by selecting relevant data sources with respect to the query, the generation of valid database queries and the consultation of the underlying databases. The system's data retrieval component consists of a set of modules interacting with the system's knowledge sources as well as a module managing the separated modules and identifying the relevant data sources for a query. Analogously to the vocabulary modules of the syntactic parser, data sources are accessed as separated modules by the main component. We will refer to components accessing data for retrieving search responses as Data Module (DM)s.

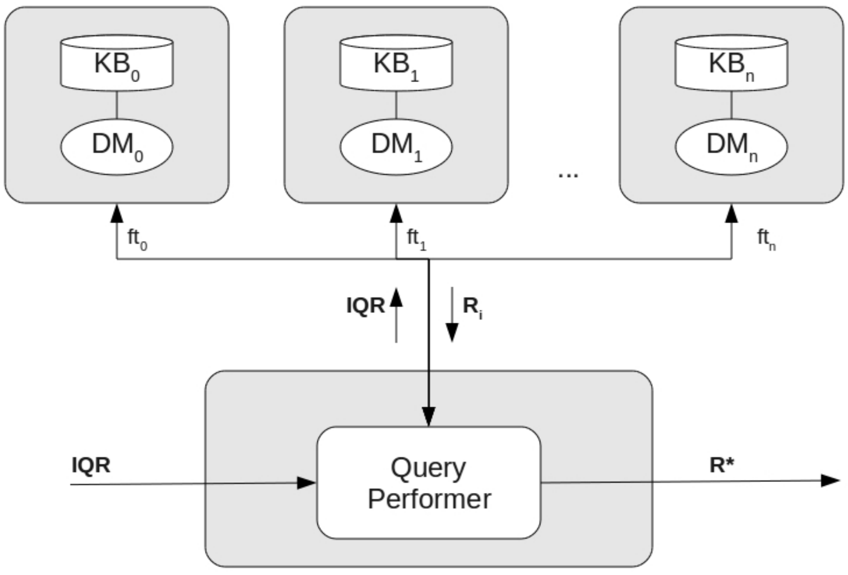


Figure 5.3: Architecture of the Query Performer with modularized access to the system's knowledge sources

Data Source Identification The purpose of the Query Performer integrated in the main system is to identify the relevant knowledge sources for a query, transmit the IQR requests to them and gather the received responses. To manage the available data modules, the system loads the configuration of each data module at startup and stores the specifications in an appropriate data structure. The query performer receives a set of intermediate representations of the semantic graphs generated by the parser, where each IQR is based on one of the registered vocabularies. The task of identifying relevant knowledge sources with respect to a query is based on two considerations:

1. Identify knowledge sources based on the request's vocabulary
2. Identify knowledge sources that are considered likely to retrieve accurate results with regard to the request

From these two assessments, 1) depicts a necessary and 2) a sufficient condition for the presented question answering system. The first statement refers to the fact that resources of the IQR are denoted with URIs. In order to generate valid database queries, data modules therefore have to be able to parse the request correctly. For this purpose, each data module is required to be specified for at least one vocabulary. The second statement aims at the goal of system effectiveness: A common problem in distributed information systems is the high computing time once the number of possible data source becomes large. In many cases not all of the available knowledge sources contain data that is relevant with respect to the given query, if the system consults each data source yet high latencies result. Additionally, if some of the knowledge sources are domain-dependent, it would be preferable only to query those that contain data relevant to the given query. To address the issue of the consideration of a knowledge source's emphasis, each data module of the system is configured with one or multiple *focus types*. A data module's focus types are a set of URIs representing the possible entities the data module can retrieve. Data module focus types correspond to the focus type identified for semantic graphs as outlined in section 4.2. A focus type represents the kind of entities the query searches for. If a data module's focus type is the empty set θ, the data module is considered domain-independent; the query performer consults the data module for each request regardless of the semantic graph's focus type. Formally, a data module d_i is employed for a request r_v of the vocabulary v if

- d_i is based on the vocabulary v
- its focus type set is the empty set θ or if
- it includes the semantic graph's focus type.

For example, a data module consulting an event data source would only be consulted if a client query searches for events, i.e. if the semantic graph's focus type is an event type such as *concert* or *festival*.

Data Modules Data Modules are components for retrieving search results from a specific data source by generating valid database queries from an IQR and transforming the result into a list of search results. In this way knowledge sources can be added independently from the database's internal organization and the storage location of the data source. Knowledge can either be stored locally on relational databases or retrieved from the web at runtime from SPARQL endpoints or APIs. They receive the IQR of the syntactic parse and retrieve a list of search results enriched with meta data. A data module generates a query for its knowledge base from the IQR received from the Query Performer. As outlined in chapter 2, depending on the knowledge base's data format a query could be a SQL query for relational databases, a SPARQL query for RDF triplestores, or an HTTP request for APIs of external applications.

5.1.3 Result Processor

The purpose of the Result Processor is to merge the result sets retrieved by the data modules to a final result set that is returned to the user interface. As the system may consult multiple data sources, the system needs to recognize results from different data sources referring to the same entity (*Instance Matching*). In order to present the most relevant search results to the client, the system finally generates a ranked result list by ordering results according to their relevance towards the user query.

To merge the result set retrieved by the employed data modules, the Result Processor iterates each result set and compares its results to results that have already been found. Redundant search results are identified if a specific set of meta data shows the same or similar values. Strong redundancy indicators are results' titles and geospatial coordinates for location-based entities. However, it is crucial only to identify two entities as congruent if all redundancy indicators are identified: For example, an event that lasts for a couple of days could be stored as a separated entity for each day, while the entries only differ in the date. A redundancy recognition would only then be correct if all indicators (title, location, date) show the same values. A local business entry, on the other hand, would be sufficiently recognized by its title and location. For this purpose, the Result Processor holds a list of redundancy indicators and compares search results according to the set of their meta data. While a general search result that has no geospatial coordinates contained in its meta data would be compared according to its title, a local business entry would be compared by its title and location.

The ranking of retrieved search results is of particular importance, as the system has to balance between the factors *relevance* of the search result and *closeness* to the client for location-based queries: On the one hand it is desirable to emphasize entities that match the client request the best. On the other hand, entities that are relevant to the query but too far away from the client's location may be considered unrelevant by the user as well. The Result Processor thus implements a stepwise model that considers both aspects: Depending on the complexity of the request underlying the retrieved results, search results are ranked according to their relevance. Within the priority computed for each request, the retrieved results are ranked according to their distance to the client location. If the number of results contained in the result set exceeds the maximum number of results requested by the client, the system only returns the most relevant results.

5.2 Interfaces

The system's modularized architecture requires the specification of various interfaces for defining the components' communication. Beside the communication between the system and external applications, the system realizes different types of communication within the internal processing of a natural language query. Based on the classification of (91), we distinguish the following types of interfaces of the system:

- **Data Structures** are transmitted between subsystems, in case of the NLI between the system's main components
- **Procedural Interfaces** define the provision of services by accessing well-defined interfaces.

Figure 5.4 depicts the realization of interfaces within the system. Data structures are employed by the main system to transmit the current stage of the query parse to the next component. Procedural interfaces are applied for two use cases: On the one hand, the system accepts requests from clients through a HTTP interface using the REST architectural style, on the other hand the system interacts through procedural interfaces with modules separated from the main system, such as vocabulary and data modules.

5.2.1 Data Structures

The following data structures are passed through the main system in the course of a request handling: The client transmits a natural language query as well as various optional meta parameters to the system's REST-compliant web service. The natural language query is then sent to the Syntactic Parser, which realizes the model presented in chapter 4 to generate a graph-based logical intermediate representation depicting the client's information request. This IQR is transmitted to the Query Performer, which generates a set of requests to the system's activated

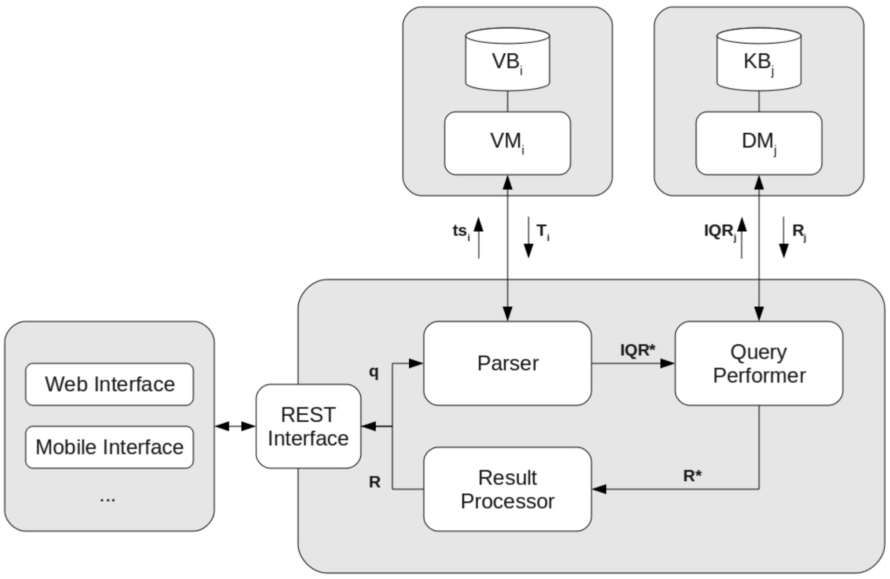


Figure 5.4: System interfaces

data modules and receives a set of search responses. The data structure passed on by the Query Performer is a set of all retrieved search results from a variable number of data modules. After merging and ranking the retrieved result sets, the Result Processor transfers the generated final result set to the web service interface.

5.2.2 Procedural Interfaces

Internally, the system communicates with two kinds of modules that are uncoupled from the main system and require well-defined interfaces: The first is the communication with the vocabulary modules to retrieve tokens from an input string. The second is the retrieval of search results from data modules.

Communication with Vocabulary Modules

The purpose of the communication of the main system and the registered vocabulary modules is the domain-independent retrieval of semantic tokens from a term sequence computed from a natural language sentence. The syntactic parser transmits the term sequence sets to all activated VMs, which compute the most probable token sequence for the given vocabulary and return the generated tokenizations. The VMs are deployed as independent components and are accessed with HTTP requests employing the REST architectural style. A HTTP VM request by the main system encodes the term sequence as a consecutively numbered list of parameters, where the parameter key is an automatically incremented number and the corresponding value is a term of the term sequence as outlined in Table 5.1.

Parameter	Value	Description
{0,1,...n}	string,max 250	Term

Table 5.1: Vocabulary Module request parameters

Listing 5.1 shows a vocabulary module request of the Tokenizer for the sample query *"universities in Koblenz"*.

```
http://espresso.uni-koblenz.de:8080/VM-DBpedia/webresources/tokens
   ?0=universities&1=in&2=koblenz
```

Listing 5.1: Vocabulary Module token request

After computing the most probable tokenization, the vocabulary modules respond the result as an XML encoded string. The tokenization consists of a probability and a list of tokens described by their unique identifier, their token type, the relevance and the underlying term. If the token type is a base token, the token is further described by a value indicating a query constraint. A VM encodes a tokenization as a list of `result`-elements representing semantic tokens, enriched with meta data such as the URI, the token type, the relevance and the query term of the particular token. Listing 5.2 presents the response of a vocabulary module.

```
<?xml version="1.0" encoding="8859-1" ?>
<resultset>

 <result>
  <id>http://dbpedia.org/ontology/University</id>
  <tokentype>c</tokentype>
  <relevance>0.99</relevance>
  <term>university</term>
 </result>

 <result>
  <id>http://dbpedia.org/resource/Koblenz</id>
  <tokentype>i</tokentype>
  <relevance>0.99</relevance>
  <term>koblenz</term>
 </result>

</resultset>
```

Listing 5.2: Vocabulary Module token response

The root element `resultset` includes a list of `result` structures representing a token each. The token's attributes are realized as sub-elements of the `result` tag.

Communication with Data Modules

The Query Performer receives an intermediate representation of the natural language query, identifies the relevant data sources according to their focus types and consults them in order to retrieve the query's search results. As described in section 4.2, the IQR consists of the request type, the focus type and a set of triples describing the query in a subject-predicate-object model. As HTTP parameters are based on a predicate-argument model, the IQR's triples have to be transmitted as values of parameters. The parameter key of each triple is, analogously to the communication with vocabulary modules, encoded as an ascending digit. Each branch of the IQR is encoded as a parameter value, whereby the triple elements are separated by commata and URIs are marked with tags. Additionally, the results' language as well as the radius for location-based requests can be specified. The request parameters are depicted in Table 5.2.

Parameter	Value	Description
e	string,max 1,[e|f]	Request type
f	string,max 250	Focus type
lang	string,max5	Language
r	int,max 50000	Radius (m)
{0,1,...n}	string	Query triples

Table 5.2: Data Module request parameters

A sample HTTP request to a data module from the query performer based on the semantic graph of the query *"universities in Koblenz"* is depicted in Listing 5.3.

```
http://espresso.uni-koblenz.de:8080/DM-SPARQL-DBpedia/webresources
    /searcher?e=e
&f=http://dbpedia.org/ontology/University
&0=?x,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    ,<http://dbpedia.org/ontology/University>
&1=?x,?y,<http://wwww.dbpedia.org/resource/Koblenz>
```

Listing 5.3: Data Module search request

The query generation and the consultation of the databases is performed by the data modules. Each data module returns an XML response containing a list of search results described with an ID indicating the rank of the result and diverse meta data such as title, description, URI, images, latitude and longitude. A sample data module response is shown in Listing 5.4.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<resultset>
 <result>
  <id>1</id>
  <meta>
   <title>Universität Koblenz-Landau</title>
   <description>Die Universität Koblenz-Landau wurde 1990 gegrü
      ndet und ist eine der jüngsten Universitäten in Deutschland.
       An ihr sind insgesamt 12.915 Studierende (Stand WS 2012/11)
       eingeschrieben, davon 6.577 in Koblenz und 6.338 in Landau
      in der Pfalz.</description>
   <uri>http://dbpedia.org/page/University_of_Koblenz_and_Landau</
      uri>
   <images>http://upload.wikimedia.org/wikipedia/commons/thumb/4/4
      b/Uni-Logo-Black-Blue.svg/200px-Uni-Logo-Black-Blue.svg.png
      </images>
   <latitude>49.204445</latitude>
   <longitude>8.108611</longitude>
  </meta>
 </result>
 <result>...</result>
</resultset>
```

Listing 5.4: Data Module search response

Communication with External Applications

External applications communicate with the system's main interface in order to provide the search functionality for user interfaces. The search interface is accessable via HTTP requests as a RESTful web service. A client request from user interfaces contains the natural language query as well as optional query-specific options, including the maximum number of results (`n`), the language of the title and description of search results (`lang`) and the client's geospatial coordinates (`lat`,`lng`). Additionally, the client may define one or multiple knowledge sources that should be consulted for a particular request (`locationtype`). For this purpose, each knowledge source is identified with an integer value used as the parameter's value for selecting a knowledge source for retrieving search results. If no `locationtype` parameter has been specified, the system consults all knowledge sources activated as defined in the system configuration. The provided parameters of the system's main interface are listed in Table 5.3.

The system's response is a list of search results that have been found for the query. It lists an identifier and various meta data for each search result such as

Parameter	Value	Description
q	string, max 250	Natural language query
n	integer: default 10, max 100	The number of results to return
lang	string, max 10: default en	Language in which the search results are displayed
lat	float: -90 to 90	Latitude of the client's location
lng	float: -180 to 180	Longitude of the client's location
locationtype	int	ID of a knowledge source

Table 5.3: System request parameters

title, description, thumbnail and the name of the result's data source. Listing 5.5 shows a possible response of the system.

```
<resultset>
 <result>
  <id>1</id>
  <meta>
   <title>Universität Koblenz-Landau</title>
   <description>Die Universität Koblenz-Landau wurde 1990 gegrü
      ndet und ist eine der jüngsten Universitäten in Deutschland.
       An ihr sind insgesamt 12.915 Studierende (Stand WS 2012/11)
       eingeschrieben, davon 6.577 in Koblenz und 6.338 in Landau
      in der Pfalz.</description>
   <uri>http://dbpedia.org/page/University_of_Koblenz_and_Landau</
      uri>
   <images>http://upload.wikimedia.org/wikipedia/commons/thumb/4/4
      b/Uni-Logo-Black-Blue.svg/200px-Uni-Logo-Black-Blue.svg.png
      </images>
   <latitude>49.204445</latitude>
   <longitude>8.108611</longitude>
  </meta>
 </result>

 <result>..</result>
 ..
</resultset>
```

Listing 5.5: Search response of the NLI

Chapter 6

Prototype

This chapter will present the system prototype, which implements the parsing model described in chapter 4 and the architecture presented in chapter 5, while fulfilling the requirements presented in chapter 3. All components of the prototype are implemented in the Java programming language[1], the lexica of the vocabulary modules have been developed based on MySQL databases[2]. Multiple data modules have been developed, gathering data on the web from SPARQL endpoints as well as APIs provided by public data providers. The communication between external applications and the natural language search system as well as the internal communication with vocabulary and data modules is based on Web services implementing the REST architectural style. On the front-end side we present the integration of the system within a mobile application as well as a web-based user interface.

6.1 Vocabulary Modules

The system's vocabularies are the major data source for associating elements of a natural language sentence with resources on the Web. Vocabulary Modules have been developed based on the DBpedia[3] (93) (94) and the LinkedGeoData[4] (96) projects, which both provide comprehensive data sets described by ontologies according to the Semantic Web standards. The data corpus employed by the DBpedia VM has been obtained from the DBpedia ontology, the DBpedia article set as well as the DBpedia category graph. The lexicon of the LinkedGeoData (LGD) VM has been generated employing the LinkedGeoData ontology as well as

[1]http://www.oracle.com/technetwork/java
[2]http://www.mysql.com
[3]http://dbpedia.org
[4]http://linkedgeodata.org

the LinkedGeoData nodes. An overview of the vocabulary modules employed by the prototype as well as their data sources is depicted in Figure 6.1.

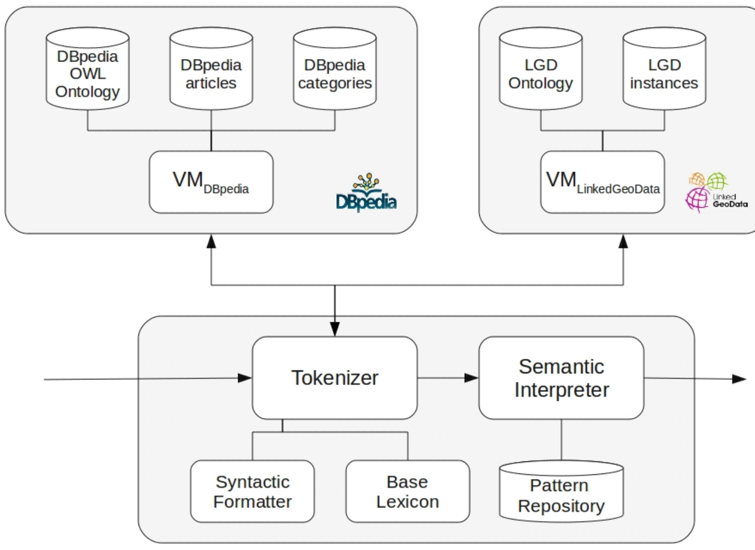


Figure 6.1: Architecture and data sources of the prototypical parser and vocabulary modules

In a first preparation, the Tokenizer employs a vocabulary-independent syntactic formatter and a base lexicon to remove punctuation marks, compute the base forms of nouns in English and German and identify base tokens. To extract classes and properties from ontologies available in Web Ontology Language (OWL), we employed the Apache Jena Framework[5] for building Semantic Web applications. For retrieving term mappings considering terminological knowledge, the labels of classes and properties of ontologies have been extracted and stored in a database associating the retrieved labels with the appropriate resources. For the prototype, class and property labels of the DBpedia and the LinkedGeoData ontologies were extracted in English and German. As the data source's instance set we employed the set of articles of the DBpedia corpus to represent instances, the article titles represent the instances' labels. Term mappings were generated as follows: Labels were separated by white spaces and added to the lexicon with the relevance according to the fraction of the label on the full entity's title. An excerpt from the lexicon mapping terms to DBpedia classes is depicted in Table 6.1.

[5]http://jena.apache.org

term	π	URI	r
sports team	c	`http://dbpedia.org/ontology/SportsTeam`	0.99
astronaut	c	`http://dbpedia.org/ontology/Astronaut`	0.99
hockey	i	`http://dbpedia.org/resource/Hockey`	0.99
operating system	p	`http://dbpedia.org/property/operatingSystem`	0.99

Table 6.1: Lexicon extract of entities extracted from the DBpedia corpus

An analogous approach has been employed for extracting term mappings from the LinkedGeoData ontology. The labels of classes and properties of the LGD ontology have been employed as term identifiers for the corresponding tokens. LGD nodes, which represent a set of points of interest described by the LGD ontology, were extracted as a set of instances. Apart from the recognition of semantic tokens within a natural language sentence based on term mappings, the recognition of base tokens is of particular importance. Base tokens represent query constraints such as specific dates or time intervals, which can be expressed in a variety of formats. Other than the association of class, role and instance tokens identified by term mappings, the recognition of base tokens is thus implemented using regular expressions. Table 6.2 shows an extract of the regular expression set for identifying a date within a German natural language query.

Format	Reg. Exp.	Example
month	`(januar)|(februar)|...`	"januar"
day-month	`([1-3][0-9]|[1-9])(\.)`	"1. januar"
	`((januar) |(februar)|...)`	
year	`([1-2][0-9][0-9][0-9])`	"2013"
day-month-year	`([1-3][0-9]|[1-9])(\.)`	"1. januar 2013"
	`((januar)|(februar)|...)`	
	`([1-2][0-9][0-9][0-9])`	

Table 6.2: Extract from the regular expression set for date recognition

When querying the lexicon for a term mapping, the vocabulary modules retrieves a list of possible tokens with varying relevances. The system additionally computes the base forms of German and English nouns and retrieves associations for them as well. If tokens of different token types feature the same relevances, the token preferences have been implemented as follows:

$$t_b > t_c > t_i > t_p \tag{6.1}$$

6.2 Data Modules

The system's DMs provide the data corpus for querying knowledge sources and retrieving search results. The modularized architecture allows data modules to be independent from the underlying DBMS as well as the data's storage location. Two types of data modules employing remote access have been developed: Data modules for knowledge sources based on SPARQL endpoints as well as data accessed through APIs. To enable the consultation of knowledge sources independently from the underlying database management system, the task of generating the database query is performed by the system's data modules. Depending on the data format, the DMs employ various methods to generate an appropriate database query from the IQR. An overview of the employed data modules is presented in Figure 6.2. Whenever a data module is added, the system generates a new directory and stores the data module's configuration in a separate file.

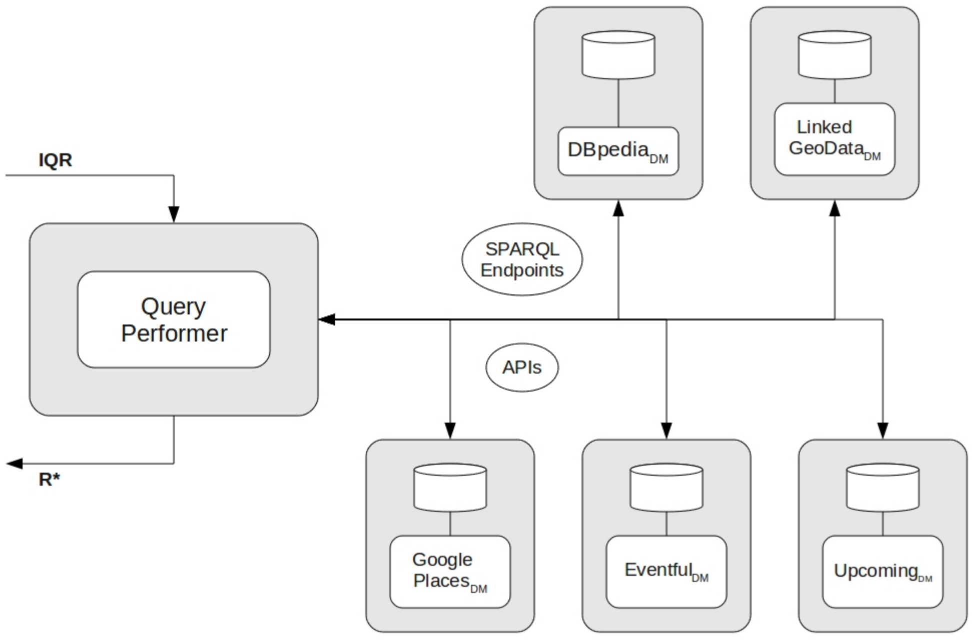


Figure 6.2: Data modules of the prototype with remote access to SPARQL endpoints and APIs

The file *config.txt* stores the Uniform Resource Locator (URL) for consulting the corresponding data module, enabling data modules to be accessed from remote as well. A sample configuration is depicted in Listing 6.1.

```
base-uri=http://espresso.uni-koblenz.de:8080/DM-API-Eventful/
   webresources
path=searcher
vocabulary=dbpedia-concept
name=Eventful
focus-type=http://dbpedia.org/resource/Category:Events (...)
```

Listing 6.1: config.txt for the data module accessing the Eventful API

6.2.1 SPARQL Endpoints

SPARQL[6] provides a graph-matching query language for RDF data and has become an official recommendation of the W3C in 2008 (69). As a public interface for the structured querying of data is one of the most practicable and most adopted ways to make data available, many LOD providers have published SPARQL endpoints[7,8] which can be accessed by external applications. By the employment of a triple-based representation of the parse and the usage of URIs for representing resources, the IQR shares many characteristics with a SPARQL query for accessing RDF repositories. In case of queries containing no filters, the IQR already represents the body of the SPARQL query. An entity request as described in chapter 4 is characterized as a query for a set of resources described by one or more constraints (e.g. *"name all universities in Koblenz"*). The SPARQL entity search includes a single variable ($?x$) in the `SELECT` clause and includes the IQR triples as the query body. In case of query constraints, an additional `FILTER` clause is added to the query. Listing 6.2 shows a sample SPARQL query generated for an entity search.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX db-owl:<http://dbpedia.org/ontology/>
PREFIX dbres:<http://dbpedia.org/resource/>

SELECT ?x WHERE {
  ?x rdf:type db-owl:University .
  ?x ?y dbres:Koblenz
}
```

Listing 6.2: SPARQL entity search

[6]http://www.w3.org/TR/rdf-sparql-query/
[7]http://dbpedia.org/sparql
[8]http://linkedgeodata.org/sparql

The presented SPARQL query retrieves a list of URIs that match the defined criteria. For providing additional data and presenting the results in a human-readable way, we defined the meta data for search result that should be provided in chapter 3. In case of SPARQL-based DMs, an additional search is necessary in order to retrieve meta data about the URIs retrieved in the first place. The query contains variables for the meta parameters *title*, *description*, *thumbnail* as well as *latitude* and *longitude* as optional parameters for location-based results. The query is a join of meta parameter queries for each of the retrieved search results. To limit the query's computing time, the data module limits the number of search results at this point of the query processing. The variables $[b, c, d, e, f]$ are employed for SPARQL meta searches. For the example query from Listing 6.2, we assume the data module querying the DBpedia SPARQL endpoint retrieved the identifiers for the University of Koblenz-Landau and the University of Applied Sciences Koblenz. Listing 6.3 shows the SPARQL meta query for the retrieved URIs.

```
PREFIX dbres:<http://dbpedia.org/resource/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xmlns:<http://xmlns.com/foaf/0.1/>
PREFIX geo:<http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT ?b ?c ?d ?e ?f WHERE { {
  dbres:University_of_Koblenz_and_Landau rdfs:label ?b .
  FILTER(langMatches(lang(?b),"EN"))
  . OPTIONAL {
    dbres:University_of_Koblenz_and_Landau rdfs:comment ?c .
    FILTER(langMatches(lang(?c),"EN")) .
    dbres:University_of_Koblenz_and_Landau xmlns:depiction ?d .
    dbres:University_of_Koblenz_and_Landau geo:lat ?e .
    dbres:University_of_Koblenz_and_Landau geo:long ?f
  }
 } UNION {
  dbres:University_of_Applied_Sciences_Koblenz rdfs:label ?b .
  FILTER(langMatches(lang(?b),"EN"))
  . OPTIONAL {
    dbres:University_of_Applied_Sciences_Koblenz rdfs:comment ?c .
    FILTER(langMatches(lang(?c),"EN")) .
    dbres:University_of_Applied_Sciences_Koblenz
      xmlns:depiction ?d .
    dbres:University_of_Applied_Sciences_Koblenz geo:lat ?e .
    dbres:University_of_Applied_Sciences_Koblenz geo:long ?f
  }
 }
}
```

Listing 6.3: SPARQL entity meta search for the query of Listing 6.2

A necessary meta parameter for search results within the SPARQL query is the retrieval of a label to display search results in a human-readable form. A description, a thumbnail image as well as geospatial coordinates are added as optional parameters. The retrieved meta data are merged with the URis retrieved in the first query and returned to the Query Performer.

In case of fact searches, the parameters for retrieving meta data are already included in the main query as additional optional parameters. A fact search retrieving the area code of Munich if shown in Listing 6.4.

```
PREFIX rdf:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX xmlns:<http://xmlns.com/foaf/0.1/>
PREFIX dbprop:<http://dbpedia.org/property/>
PREFIX dbres:<http://dbpedia.org/resource/>

SELECT ?x ?b ?d ?f WHERE {
 dbres:Munich dbprop:vorwahl ?x .

 OPTIONAL {
  dbres:Munich rdf:label ?b .
  dbres:Munich xmlns:depiction ?d .
  dbprop:vorwahl rdf:label ?f
  FILTER(langMatches(lang(?b),"DE")) .
 }
}
```

Listing 6.4: SPARQL fact search

The SPARQL query returns the value of the requested property, the property's label as well as a label and an image of the related entity as optional parameters. For example, a client transmitting a German query requesting the area code of Munich (*"vorwahl von münchen"*) would receive the property label (*"Vorwahl"*), the property value (*"89"*), as well as the label (*"München"*) and an image of Munich. The response of a fact search in the web frontend is depicted in Figure 6.3. The frontend displays the title and a thumbnail of the particular entity as well as the fact label and the requested value.

If a client search is focused on a specific area (e.g. *"universities in Koblenz"*) or if geospatial coordinates indicating the client's location are transmitted with the request, the SPARQL query is specified by defining a query around a circular area. If the request additionally contains a *radius* parameter, the SPARQL query is specified to search within an area of the defined radius. Listing 6.5 shows a circular search around a point of interest for a NL query *"bakeries"* in Koblenz with a radius of 2 kilometres employing the LinkedGeoData vocabulary. Geospatial coordinates of points of interest such as city names are identified by an additional location-based lexicon containing the geospatial coordinates of location-based entities such as cities and points of interests. The geospatial coordinates have been extracted

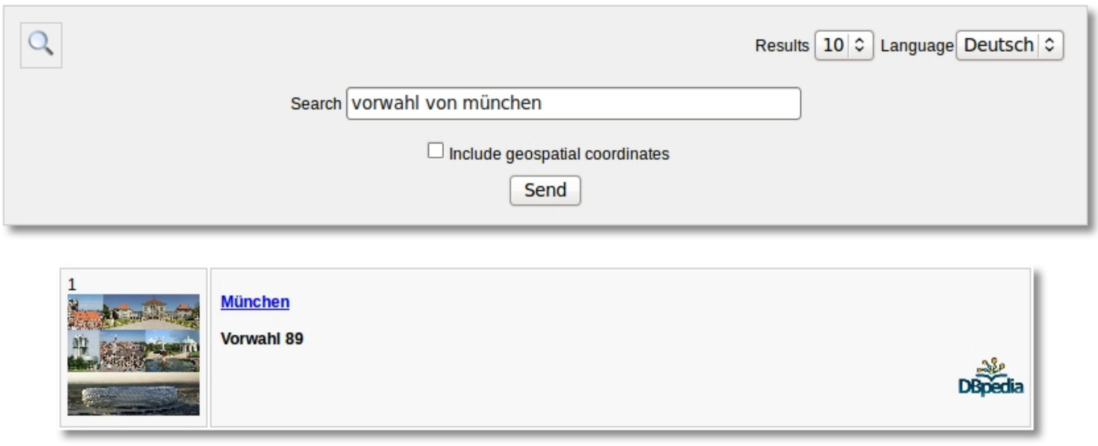


Figure 6.3: Search response of a fact search within the Web frontend

from the LinkedGeoData data corpus. In this way, phrases refering to location-based entities (e.g. *"Koblenz"*) are associated with their geospatial coordinates and employed as the geographic basis for the query processing. In this way, a query *"universities in Munich"* would identify the coordinates of Munich and perform a circular search for universities with the Munich city centre as the query's central location.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX pos:<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX lgd:<http://linkedgeodata.org/ontology/>

SELECT ?x WHERE {
 ?x rdf:type lgd:Bakery  .
 ?x pos:geometry  ?xgeo .

 FILTER ( <bif:st_intersects> (?xgeo, <bif:st_point>
    (7.5951959,50.3511528),2.0) )

}
LIMIT 15
```

Listing 6.5: Location-based SPARQL entity search

6.2.2 Application Programming Interfaces

The employment of data sources consulted by application programming interfaces as knowledge sources differs significantly from data modules based on relational databases or RDF repositories. APIs are often consulted with parameters in a predicate-argument format and generally have their own parameters which are not based on vocabularies published in a structured way. We therefore can neither address resources with URIs nor use the triple format of the IQR directly for the query. However, we can employ APIs by creating a mapping between elements of a registered vocabulary and API parameters. If we assume a request's parameters to be of a predicate-argument form, a parameter mapping associates elements of a vocabulary such as a property or a class with parameters included in an API request. In order to map an IQR triple to a predicate-argument structure of an API request, we can make the following implications for triples of IQR: 1) a triple may only contain 1 variable, which is as well the focus variable, 2) a triple must be of the following form: [?variable]-predicate-argument. A parameter mapping $pm(e, p)$ maps an element of a vocabulary to a parameter of an API request, which may be interpreted as follows: If a triple of an IQR contains a URI u as its predicate and there exists a mapping $(u \to p)$ that assigns u to an API parameter p, the parameter p and the triple's object value form a query assignment. If there exists a mapping $(u_p \to p)$ and a mapping $(u_o \to v)$ that assigns a URI of a triple predicate to an API parameter and a URI of a triple object to an API parameter value, the duple (p, v) is generated as a query assignment.

An example of a mapping between the intermediate representation of a natural language query and the API of an external application is read as follows: The local events discovery and demand service *Eventful*[9] provides an API with various parameters for searching events, including the specification of the location and date of an event. Additionally, clients can specify a category of the requested events (*music*,*performing arts*, etc.). The sample input query "concerts in january" results in the IQR query triples shown in Listing 6.6.

```
PREFIX db-category:<http://dbpedia.org/Category:>
PREFIX nlsearch:<espresso.uni-koblenz.de#>
{
  ?x nlsearch:category db-category:Concerts .
  ?x nlsearch:date "2013010100-2013013100"
}
```

Listing 6.6: IQR triples generated for the query *"concerts in january"*

A mapping between a base token and the Eventful API assigns a base token *date* to the Eventful parameter *date* specifing a time interval for an event search. A query

[9]http://www.eventful.com

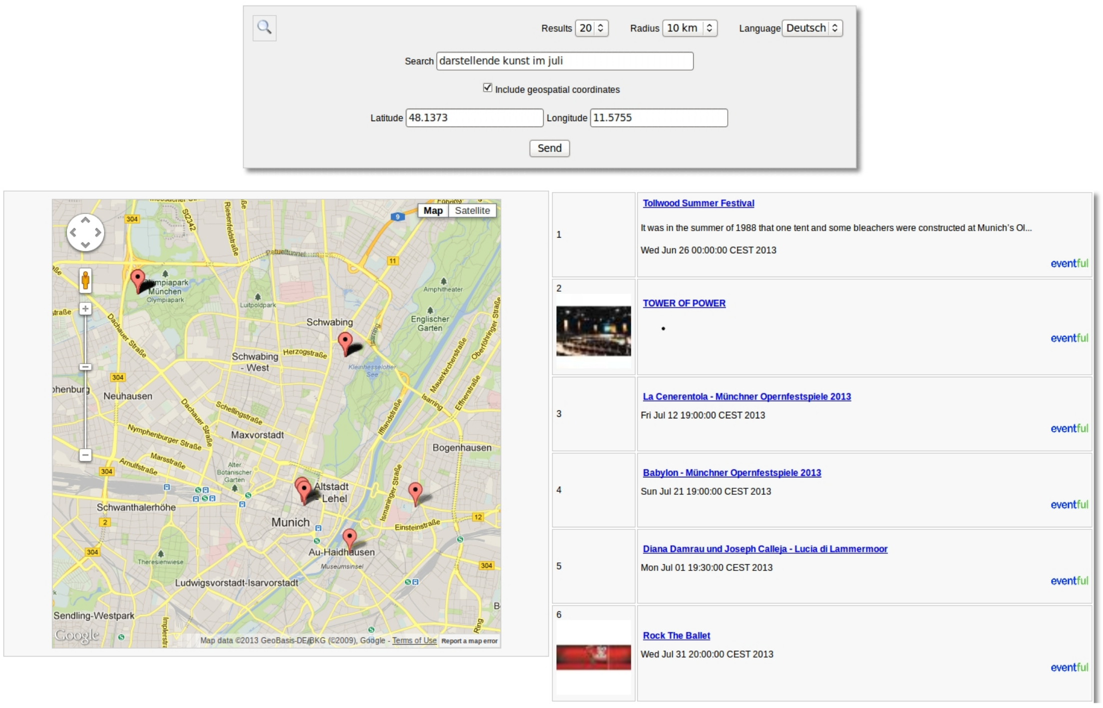
"concerts in january" would thus result in a triple specifying the time interval as a triple of the form (?x - date - 2013010100-2013013100). If the system applies the mapping, the event search would include the parameter *(date=2013010100-2013013100)* as a parameter within the HTTP request. A mapping between the DBpedia ontology and the Eventful API assigns DBpedia categories to the category parameter of the Eventful API. As the Eventful API has a custom list of available categories, the parameter values have to be identified with a mapping as well. An excerpt of a corresponding value mapping is shown in Table 6.3. A search response

DBpedia category	Eventful category
db-cat:Concerts	event:music
db-cat:Information_technology	event:technology

Table 6.3: Excerpt of the mapping between DBpedia categories and categories of the Eventful API

of a query using specified parameters displayed in the web frontend is depicted in Figure 6.4.

An additional feature that was implemented for retrieving precise location-based search results is the identification and search for location-based related terms such as related local businesses or points of interest. A common problem of searching for a particular product is the specification of appropriate search terms: If a product may be denoted in several ways or the user searches for a group of products with a specific feature (e.g. products of a specific manufacturer), the user may have to perform multiple searches. Especially if products may be available in stores of various kinds, the precise formulation of a query may be difficult. For example, a user that searches places to buy smartphones could search for the term "phone shop". However, it may also be possible that computer stores or electronics store may offer the product the user searches for. A more comfortable way for users would be if they could simply enter the product they are searching for and receive proposals of possible locations where the appropriate product can be acquired. For this purpose, the system needs to know where to find the products the users search for, i.e. requires a mapping of entities to related location-based terms. Within the scope of the prototypical implementation, two layers of location-based mappings have been generated: The first is a hand-crafted, precise mapping that defines points of interest related to a particular term (e.g. *spaghetti* $\rightarrow$ *italian restaurant*). This mapping ensures a high degree of precision and consistency of the retrieved associations. However, since the generation of hand-crafted assignments is an expensive and time-consuming task, a second, more farreaching mapping has been generated additionally. The second mapping is an automatically generated extensive mapping that associates real-world entities with related location-based



Results 20 Radius 10 km Language Deutsch

Search darstellende kunst im juli

☑ Include geospatial coordinates

Latitude 48.1373 Longitude 11.5755

Send

1		Tollwood Summer Festival It was in the summer of 1988 that one tent and some bleachers were constructed at Munich's Ol... Wed Jun 26 00:00:00 CEST 2013 eventful
2		TOWER OF POWER • eventful
3		La Cenerentola - Münchner Opernfestspiele 2013 Fri Jul 12 19:00:00 CEST 2013 eventful
4		Babylon - Münchner Opernfestspiele 2013 Sun Jul 21 19:00:00 CEST 2013 eventful
5		Diana Damrau und Joseph Calleja - Lucia di Lammermoor Mon Jul 01 19:30:00 CEST 2013 eventful
6		Rock The Ballet Wed Jul 31 20:00:00 CEST 2013 eventful

Figure 6.4: Search response of a search using specified parameters within the Web frontend

points of interest. Formally, a location-based mapping $m(w_a \rightarrow w_l)$ is a mapping that recursively assigns concepts of a hierarchical world model w_a to a set of location-based concepts w_l. A popular representation of a small-world hierarchical structure is the Wikipedia Category Graph, which was employed to extract the sets of possible products. As the base for location-based terms the classes of the LinkedGeoData ontology were employed. The developed location-based mapping recursively maps Wikipedia categories including all their particular sub-categories to LinkedGeoData classes. A sample location-based mapping could be realized as follows: We assume there is a Wikipedia category *Electronics* that has the sub-categories *"Mobile device"* and *"Hardware"*, and there exists a class within the LinkedGeoData ontology *"electronics store"*. A location-based mapping could be of the following form:

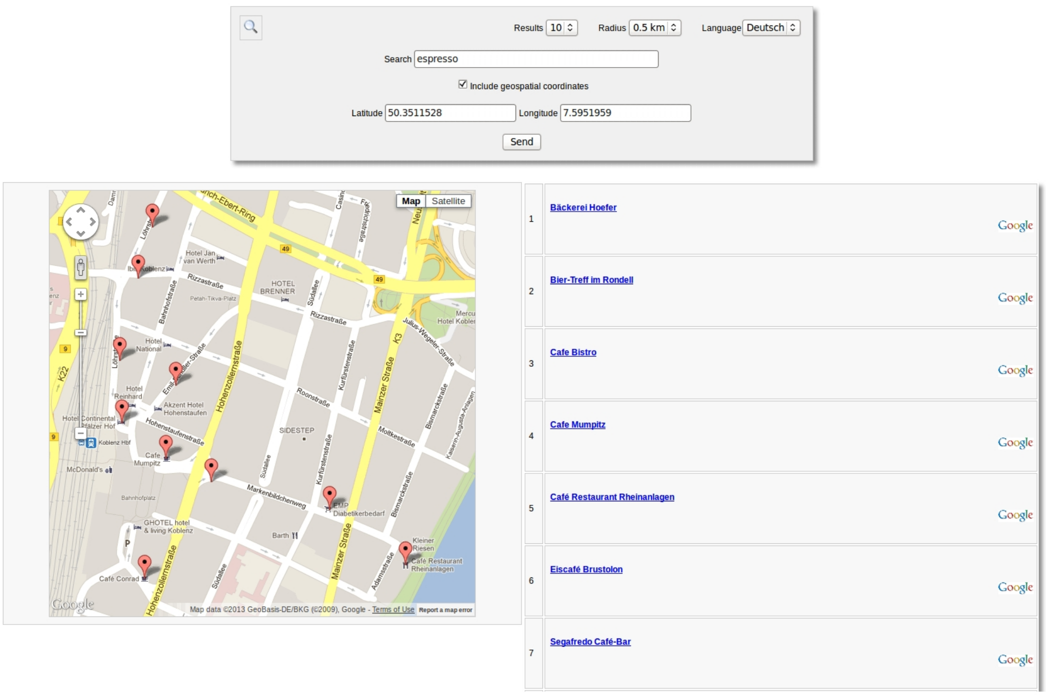
$$dbcat : Electronics \rightarrow lgd : electronics\_store$$

As the method recursively associates all subcategories of dbcat:Electronics with lgd:Electronics_store, the mapping generated would thus be of the form

$$dbcat : Electronics \rightarrow lgd : Electronics\_store$$

$$dbcat : Mobile\_devices \rightarrow lgd : Electronics\_store$$
$$dbcat : Hardware \rightarrow lgd : Electronics\_store$$

The mapping assigns the Wikipedia category and all of its subcategories to a LinkedGeoData class. Additionally, instances that belong to the particular categories have been extracted and included in the mapping as well. In this way the system is able to parse queries for specific products or points of interest. If the system receives a client request including the natural language query *"mobile device"* or a specific device, the system automatically will also search for the related location-based concept. Figure C.4 shows a set of location-based related LGD classes as well as a location-based related term retrieved for the generated focus type of the query. If the client transmits geospatial coordinates as meta parameters in the request, the system performs additional location-based searches for each additional location-based term. Since results of additional searches based on the hand-crafted mapping are likely to be very precise, they receive the highest ranks from the Result Processor when generating the query result set. A possible location-based search within the Web frontend and the display of the retrieved results is depicted in Figure 6.5.


Results 10 Radius 0.5 km Language Deutsch

Search espresso

☑ Include geospatial coordinates

Latitude 50.3511528 Longitude 7.5951959

Send

1	Bäckerei Hoefer Google
2	Bier-Treff im Rondell Google
3	Cafe Bistro Google
4	Cafe Mumpitz Google
5	Café Restaurant Rheinanlagen Google
6	Eiscafé Brustolon Google
7	Segafredo Café-Bar Google

Map
Satellite
Hohenzollernstraße
Mainzer Straße
Map data ©2013 GeoBasis-DE/BKG (©2009), Google - Terms of Use Report a map error

Figure 6.5: Search response of a location-based search within the Web frontend

6.3 User Interfaces

As the NLI communicates with external applications through a well-defined interface, it is not attached to one particular system and can be accessed by multiple applications independently from the client's location and system. The user interface provides users an input form for entering a natural language query and presents the retrieved search response in a customized way. For the scope of the system's prototype, there have been developed a mobile interface accessing the system from within a mobile application as well as a browser-based web interface.

6.3.1 Mobile Interface - Integration in the NAPA Pedestrian Navigation System

A mobile access to the system has been realized by integrating a query module within a mobile application, enriching an existing application by a natural language search functionality. The base of the mobile system is the research project NAPA[10] (Navigationsempfänger Chipsatz für Personennavigation mit Anwendungen bei erhöhter Genauigkeit)(97), which has developed the technical foundations for precise pedestrian navigation. For this purpose, NAPA provides a high degree of precision in the fields of location determination and navigation, taking into account pedestrian crossings, traffic lights, subways and crossovers (98). The technologies developed in the project have provided the foundations for a variety of possible areas of application, such as pedestrian navigation, navigation solutions for wheelchair users or visually impaired persons, driving assistance and automated emergency call systems (97). A variety of institutions such as industrial companies, small and medium-sized businesses, research facilities and universities have contributed to the development of NAPA, including IMST, NAVIGON, NAVTEQ, Fraunhofer-IIS, NavCert, RWTH Aachen as well as the University of Koblenz and Landau (97). The provision of a precise functionality for specialized search and discovery of points of interest is of particular importance for a navigation system. By coupling the NLI with the pedestrian navigation system, the existing search functionality of NAPA is enriched by the ability to parse complex queries as well as full natural language sentences. The NLI is able to parse questions including specified parameters as well as to associate the query with local businesses or institutions.

Within the scope of this thesis, the functionality for accessing the NLI and processing the result list returned by the system has been provided. The mergence of search results of the NLI and the existing search functionality, the display of search results as well as the enrichment of search results with further meta data

[10]http://projekt-napa.de

has been realized by the NAPA development team. The communication between the mobile interface and the NLI proceeds as defined in section 5.2. The mobile interface consults the system via HTTP request, transmitting a natural language query and diverse meta parameters such as the query's radius and the client's location. The precise location determination by mobile devices enables the automated transmission of the client's location and a precise pedestrian navigation to location-based results retrieved by the system. Users may additionally define the radius and the knowledge sources that are consulted for retrieving points of interest in the client's close proximity within the graphical user interface of the mobile application. The employed IDs for identifying knowledge sources by the NAPA system and the NLI system have been synchronized: When consulting the NLI system, the IDs of activated knowledge sources are added additionally as request parameters by the NAPA system. In this way, the NLI system as well as the existing search functionality retrieve search results from the knowledge sources defined by the client. A sample request of the NAPA system that searches for universities within a radius of 2 kilometres consulting 2 specified knowledge sources is shown in Listing 6.7.

```
http://espresso.uni-koblenz.de:8080/NaturalLanguageSearch/
    webresources/search?q=universities+in+koblenz
    &r=2000&lat=50.3511528&lng=7.5951959
    &locationtype=0&locationtype=2
```

Listing 6.7: Search request sent by the NAPA system

The response retrieved by the NLI corresponds to the sample response Listing 5.5. The presentation of search results retrieved by the NLI within the mobile application is shown in Figure 6.6 and 6.7 provided by the NAPA development team. Figure 6.6 depicts the map-based display of entities that have been retrieved for a circular search around the client's current location. Search results retrieved by the NLI are displayed with specified markers to distinguish them from results retrieved by the existing search functionality. Each result may be selected by the user, followed by a detailed presentation of the particular result as shown in Figure 6.7. Search results are enriched with further meta data by the NAPA system: In Figure 6.7, the search result is enriched with data concerning the address and phone number of the point of interest.

6.3.2 Web Interface

The web interface for accessing the system prototype presents the natural language search system functionality within a web browser by offering a Hypertext Markup Language (HTML) input form to the user and presenting the retrieved response

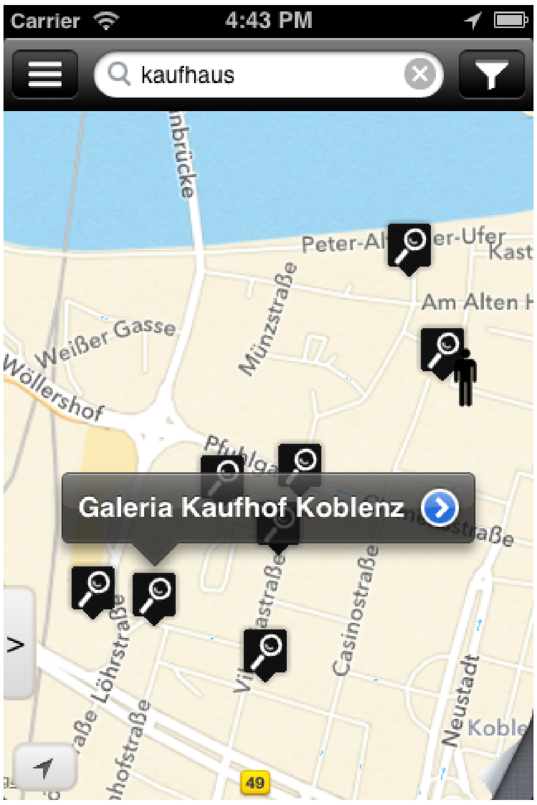
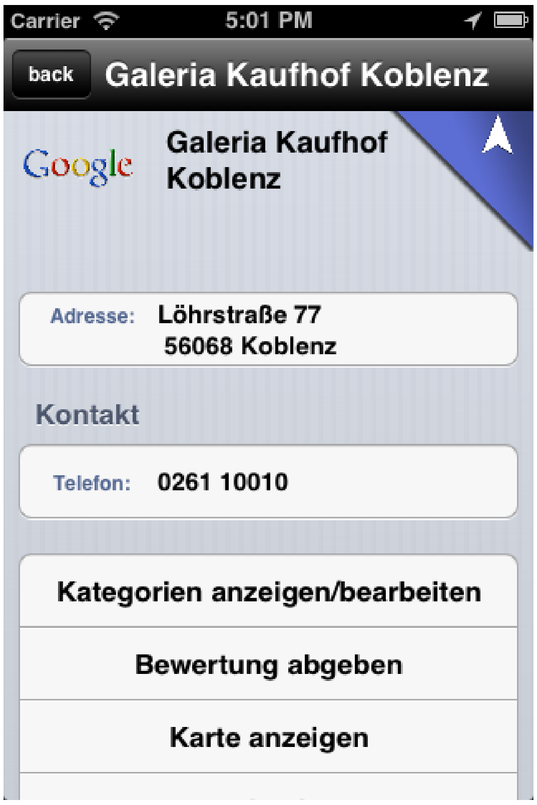


Figure 6.6: Display of search results within the NAPA mobile application (5)


Carrier 5:01 PM

back Galeria Kaufhof Koblenz

Galeria Kaufhof Koblenz

Adresse: Löhrstraße 77
56068 Koblenz

Kontakt

Telefon: 0261 10010

Kategorien anzeigen/bearbeiten

Bewertung abgeben

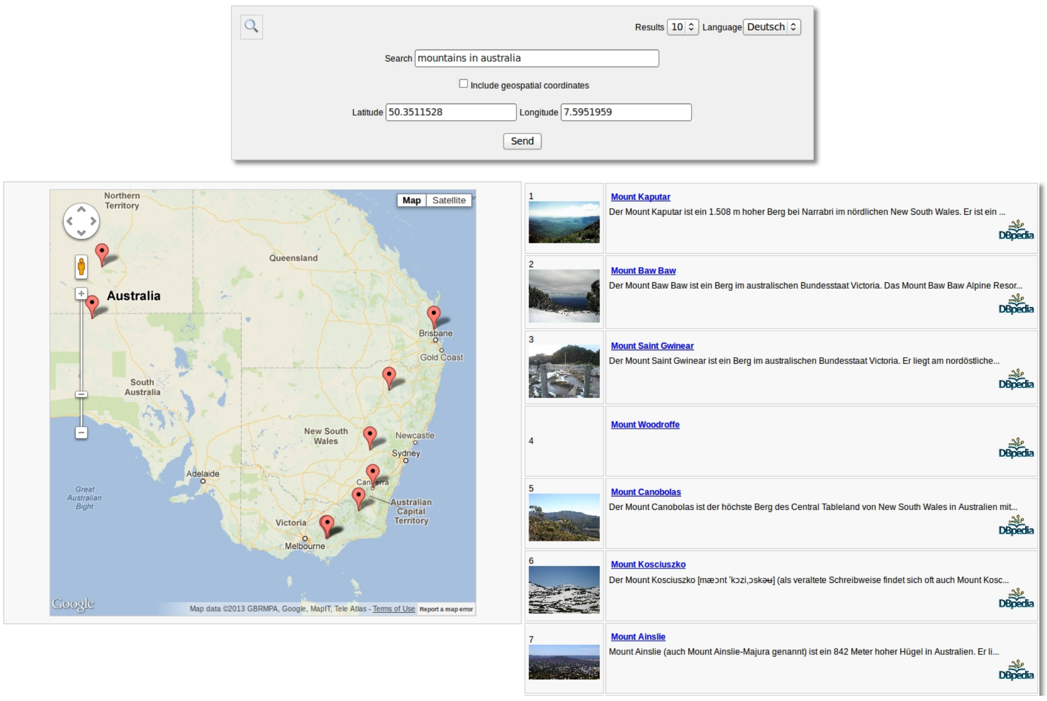
Karte anzeigen

Figure 6.7: Display of search result meta data in NAPA (5)

from the server as defined in section 3.1. It provides the possibility to search both for general and local information by optionally activating the transmission of geospatial coordinates. If the NLI receives geospatial coordinates included in the query parameters, the system computes a location-based search for points of interest located near the client's location. The input form further provides the ability to define the radius of the search, the maximum number of results as well as the language of the retrieved results. Entities are displayed as a list providing the title, a description, an image and the data source of each resource. As entities are identified by their URI, users can retrieve more information about the resource by following the title's hyperlink. The knowledge source of each search result is mentioned as well; if a thumbnail of the knowledge source's logo is available, the knowledge source is represented as an image. Additionally the system can provide a map-based display for location-based retrieved entities. To display items described by geospatial coordinates on a map, the Google Maps JavaScript API Version 3[11]

[11] https://developers.google.com/maps/documentation/javascript/
last visited on: May 31th, 2013.

has been employed for integrating a customized map displaying the results of the client request within the web interface. If meta data about latitude and longitude of a resource are found in one or more search results, the system provides a map display presenting all geospatial results. To associate an element displayed on the map with an entry included in the result list next to the map, the rank of each result has been added as the result's title for each marker on the map. Next to the map, the result list is displayed to provide additional information about the resources. A search for general entities using the Web frontend displaying the retrieved results on a map as well as a list is shown in Figure 6.8. Facts of a specific entity are displayed together with the entity they belong to. The system presents the title and a thumbnail of the entity and places the retrieved property and its value beneath. For the purpose of analyzing and testing the semantic graph computation, a user interface consulting the Syntactic Parser has been developed. It provides the possibility to enter a natural language query and to define a vocabulary module, the system then computes the tokenization and the semantic graph of the query for the defined VM as shown in the Figures C.1 C.2 C.3.


Results 10 Language Deutsch

Search mountains in australia

☐ Include geospatial coordinates

Latitude 50.3511528 Longitude 7.5951959

1 **Mount Kaputar**
Der Mount Kaputar ist ein 1.508 m hoher Berg bei Narrabri im nördlichen New South Wales. Er ist ein ...

2 **Mount Baw Baw**
Der Mount Baw Baw ist ein Berg im australischen Bundesstaat Victoria. Das Mount Baw Baw Alpine Resor...

3 **Mount Saint Gwinear**
Der Mount Saint Gwinear ist ein Berg im australischen Bundesstaat Victoria. Er liegt am nordöstliche...

4 **Mount Woodroffe**

5 **Mount Canobolas**
Der Mount Canobolas ist der höchste Berg des Central Tableland von New South Wales in Australien mit...

6 **Mount Kosciuszko**
Der Mount Kosciuszko [mæɔnt ˈkɔzi͵ɔskəʉ] (als veraltete Schreibweise findet sich oft auch Mount Kosc...

7 **Mount Ainslie**
Mount Ainslie (auch Mount Ainslie-Majura genannt) ist ein 842 Meter hoher Hügel in Australien. Er li...

Figure 6.8: Search results of a general entity search displayed in the Web frontend

Chapter 7

Evaluation

The drawing of profound conclusions considering the performance of the approach developed in this thesis requires an evaluation of the system. This chapter presents the evaluation that has been performed on the system prototype presented in chapter 6 in order to validate the system's performance. First we will outline the goals of the evaluation, followed by a description of the evaluation metrics and the test cases generated for the evaluation. The evaluation comprises of the measurement of the syntactic parser's accuracy as well as the computation of the precision and recall of the search results retrieved by the system. Additionally, the processing speed of the system as well as each component has been measured. Finally, we will discuss the strengths and weaknesses of the system's components and give an outlook for further development of the system.

7.1 Goals

In order to verify the system, the evaluation focuses on two aspects of the system: 1) Whether the system is able to parse natural language queries correctly into a logical intermediate form, and 2) whether the retrieved search results are relevant with respect to the query and if all available relevant items have been found. Additionally, the required processing time of the system as well as the time required by each system component in order to compute a query is of interest. The evaluation is thus comprised of 3 phases: The analysis of the system performance in terms of accuracy and scope of the search engine as well as the measurement of the system's processing time. The performance measurement of the system aims at analyzing the system's components to make conclusions about the accuracy of each module of the prototype as well as the relevance of the retrieved data. The performance measurement comprises of the measurement of the following aspects:

1. **Accuracy:** Measurement of the correctly parsed queries of the syntactic parser as well as each parsing component

2. **Scope:** Evaluation of precision and recall of the retrieved search results

The first part of the performance evaluation will supply data about the correctness and precision of the system in terms of its capability to model the information request of the client. The evaluation of the scope of the system concentrates on analyzing the responses retrieved by the prototype, in particular precision and recall of the result set. In this way we will gather data about the relevance of the retrieved responses and the performance of the Result Processor constructing the result set. Within this context, we will be able to draw conclusions whether the queries generated by the data modules may have been formulated too narrow or too general. The second goal of the evaluation is the measurement of the processing time of the search system and each component. The required time of the syntactic parsing procedure will provide information about the internal processing of different test case questions. By measuring the time required by the system's vocabulary modules, we will be able to estimate the correlation between the size of the queried databases and the time required for retrieving a query's tokenization. Furthermore, the processing time of the system's data modules will provide data about the time required by components based on APIs as well as SPARQL endpoints as knowledge sources.

7.2 Methodology

The evaluation metrics employed for the prototype testing depend on the purpose of the particular test unit and the output produced by the component. This chapter will provide an overview of the evaluation metrics employed for the system evaluation as well as the design of the test cases.

7.2.1 Evaluation Metrics

The performance testing of the prototype consists of an accuracy measurement of the syntactic parser as well as an evaluation of the system's scope. An introduction to various aspects of the evaluation of NL systems is presented by (99), such as the distinction of intrinsic and extrinsic evaluation methods as well as component and end-to-end tests. In this context, the system evaluation provides both perspectives: While the accuracy measurement is designed as an intrinsic component test evaluating each system component's performance separately, the system scope evaluation is based on an extrinsic end-to-end evaluation, providing a randomly selected input query to the system and analyzing the system's response from an external view. Our first goal is the measurement of the system's accuracy. To evaluate the performance of a system that generates one output per input, (99) proposes to employ the percentage of agreement with the output produced by the system and a set of human-annotated 'ground truth' sense labels, i.e. the accuracy, as the primary evaluation metric:

$$A = \frac{\sum_{i=1..n} agr_i}{n} = \frac{number\ correct}{n} \tag{7.1}$$

where agr_i is 1 if the system output corresponds with the intended test case output and 0 otherwise (99) (100). Applied to the task of generating a natural language query tokenization, the component's accuracy can be measured by computing the percentage of correctly retrieved tokens. The system scope evaluation consists of the analysis of the completeness of the system response as well as the relevance of the retrieved search results. Other than the accuracy measurement of the system components, the evaluation of the search results retrieved by the system often cannot be measured by comparing the output to a single correct response. In case of multiple outputs per input, (99) makes the following assumptions: Either an entity is retrieved by the system or it is not, and either it is relevant to the information requested or it is not. This abstraction makes it possible to define the quality of a system's output with the concepts precision and recall. (99) defines Precision as the fraction of system output that is relevant, i.e.

$$P = \frac{r}{r+n} \tag{7.2}$$

where r counts the number of documents that are relevant and retrieved by the system, and $(r+n)$ is the total number of documents (99). Recall as defined by (99) (101) is the fraction of relevant documents that is retrieved in relation to all possible relevant documents, i.e.

$$R = \frac{r}{r+T} \tag{7.3}$$

where T represents the relevant documents that have not been retrieved by the system, that is, $(r+T)$ is the total number of documents that should have been retrieved (99) (101). The retrieval of a high precision rate often comes at the cost of recall, e.g. when a system returns only a few results with a high relevance rate (99). A metric that balances both precision and recall depending on the size of the result set r is obtained with the harmonic mean $HM(r)$ (101):

$$HM(r) = \frac{2}{\frac{1}{R(r)} + \frac{1}{P(r)}} \tag{7.4}$$

7.2.2 Test Cases

(91) identifies four steps in the process of software testing: 1) Designing of the test cases, 2) the generation of a test corpus, 3) execution of the program with test data and 4) the comparison of the results with test cases. We applied this concept to measure the accuracy of the prototype's syntactic parser by generating a set of possible natural language queries as well as their intended parses. The evaluation of the system accuracy is based on the comparison between the semantic graph generated by the syntactic parser and an exemplary graph provided by a test case, representing the intended intermediate query representation of a natural language request. The procedure is designed as an automated process, randomly selecting a test case from a corpus, parsing the query and comparing the results generated by the system with the exemplary result provided by the test case. Each test case consists of a natural language query and the intended correct graph-based intermediate representation, comprising of the query's tokenization as well as the correct relation and focus pattern employed for parsing the query. The test cases generated for the system evaluation represent a set of exemplary requests to the system, providing both a sample query as well as the output the system should ideally provide. Analogously to the various types of questions the prototype may answer presented in chapter 6, the test corpus consists of 5 different query types providing location-based and general questions:

- Entity requests (local) (LE)
- Local business requests (local) (LB)
- Specific parameter requests (local) (LSP)
- Entity requests with a resource constraint (general) (GE)
- Fact requests of an entity (general) (GF)

For the purpose of accuracy measurement of the syntactic parser, each test case has been generated with an exemplary intermediate stage providing the favoured parsing of the query, consisting of the retrieved tokens as well as the relation and focus patterns employed for generating the semantic graph. A set of sample test cases as well as the intended tokenizations is displayed in Table 7.1, where the tokens are depicted as token types *[c,p,i,b]* as introduced in section 4.1. Requests searching for location-based entities have been modeled as test cases comprising of the labels of location-based concepts as queries and geospatial coordinates representing the geographical basis for the query. Test cases implementing queries including specified parameters have been generated for a customized event search including a specific event type and a date interval. The retrieval of local businesses for products or other real-world entities is based on the mappings between

Type	NL query	Tokenization
GE	"universities in Koblenz"	c,i
GF	"location of Burg Eltz"	p,i
LE	"bakeries"	c
LB	"pretzels"	i
LSP	"festivals in may"	c,b

Table 7.1: Examples of test case queries of different question types

real-world concepts and location-based concepts as described in section 6.2. To evaluate the correct retrieval of location-based classes, test cases for local business requests have been extended with a location-based concept that is intended to be associated with the actual entity. For the purpose of identifying real-world entities that are likely to be acquirable in close proximity, instances of DBpedia categories which have been associated with LinkedGeoData classes have been retrieved. The instances labels of the retrieved instances have been employed as possible keywords for test case queries. For generating test cases for general-purpose requests, a set of classes from the DBpedia ontology has been selected and entities belonging to these classes have been retrieved via querying the DBpedia SPARQL endpoint. General questions have then been generated by combining certain classes of the DBpedia ontology with randomly selected entities of another selected class. For example, a possible general entitiy query would be to search for universities in a particular city, i.e. a test case query of the form *class(dbpedia:university) + instance(dbpedia:city)*. A possible query would thus be *"Universities in Koblenz"*. A sample SPARQL query to retrieve a list of directors from the DBpedia corpus is displayed in Listing 7.1.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdf-schema:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia-owl:<http://dbpedia.org/ontology/>
PREFIX dbres:<http://dbpedia.org/resource/>

SELECT DISTINCT ?x ?y WHERE {
  ?x rdf:type dbpedia-owl:Person .
  ?y ?z dbres:Film_director .
  ?x rdf-schema:label ?y .
  FILTER (langMatches(lang(?y),"DE"))
}
```

Listing 7.1: Sample SPARQL query to retrieve entities for test cases

For the purpose of generating a set of test cases including fact queries, a set of class-specific properties has been retrieved and combined with instances of the particular class (e.g. fact search for class `dbpedia:Automobile`: *"manufacturer* *of VW Golf"*). To measure the recall of the system responses, exemplary results have been retrieved from the publicly available interfaces of the employed data sources.

7.3 Results

The evaluation has been performed with a machine employing an AMD Athlon 64 X2 Dual Core Processor 3800+ with approx. 3262 MB of Random Access Memory (RAM). For each of the possible 5 request types listed in section 7.2, the prototype has processed a set of test cases and evaluated the results. Additionally, tests were performed for test cases in German and English. The evaluation has been performed with 3 active vocabulary modules, namely VMs accessing the DBpedia (*dbpedia*) and LinkedGeoData (*linkedgeodata*) ontologies, as well as a VM accessing the DBpedia ontology including the DBpedia category graph as an additional resource pool (*dbpedia-concept*). On the knowledge retrieval side, the following data modules have been activated: Two DMs querying the SPARQL endpoints of the DBpedia (*sparql-dbpedia*) and LinkedGeoData (*sparql-linkedgeodata*) corpora, as well as three DMs accessing the APIs of GooglePlaces (*api-googleplaces*) and the event platforms Eventful (*api-eventful*) and Upcoming (*api-upcoming*).

7.3.1 System Accuracy

The system accuracy has been measured as the percentage of test cases that have been parsed correctly by the Tokenizer and the Semantic Interpreter. The tokenization accuracy has been measured as the fraction of correctly identified tokens, the semantic interpretation has been marked as correct if the parser employed the correct relation pattern as well as the correct focus pattern for the semantic graph generation. An accuracy test was performed by an algorithm randomly selecting a test case of a specific request type from the database, processing the query like a client request and evaluating the retrieved results. For each of the question types presented in section 7.2, the algorithm has randomly selected 40 test cases in German and English and compared the result with the semantic graph defined by the test case. Altogether, the parser accuracy has thus been tested on 400 test cases. Fig. 7.1 shows the average accuracies computed by the syntactic parser.

All tokenization and semantic interpretation accuracies have ranged between an accuracy of 70-100 percent. The retrieved results show that the parser tends to compute with a higher accuracy for requests for location-based entities, which may be attributed to a higher complexity of natural language queries concerning general information requests in the test corpus. Especially in case of test cases including fact searches, the identification of instances described by a plurality of

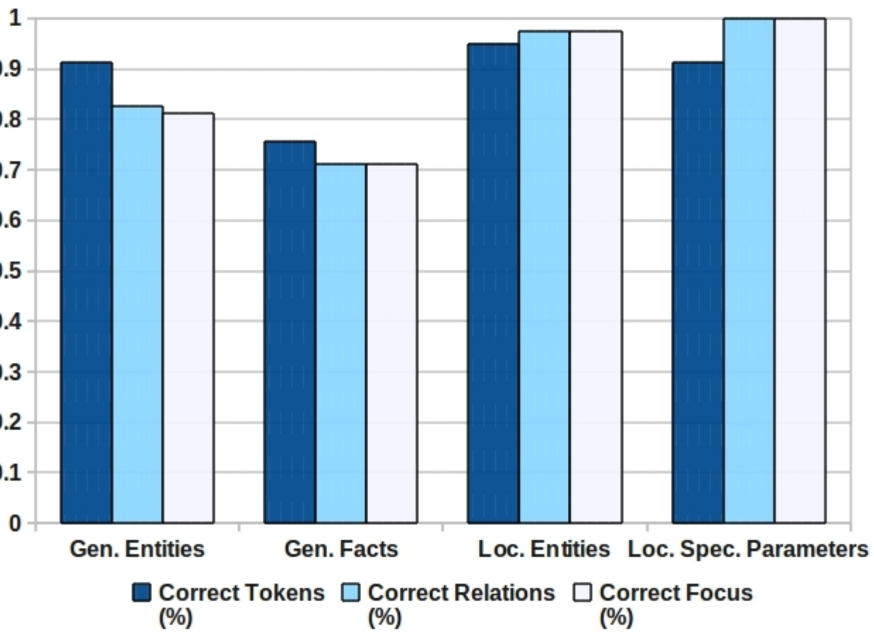


Figure 7.1: Average parsing accuracies of the prototype's query types

words has turned out as a challenge. The average accuracies of the system's components parsing German (DE) and English (EN) test cases are listed in Table 7.2[1], consisting of the fractions of correctly parsed tokens, graph relations and the focus. Queries including location-based references have generally reached higher accuracy values than general questions, which may indicate ambiguities in general queries as one of the major challenges of the parsing process. Remarkable is the employment of correct relation and focus patterns in case of test cases including local specified parameters. All query results have shown a remarkably high consistency of the focus discovery: In 99.75% of all cases where the relations have been computed correctly, the Semantic Interpreter also employed the correct focus pattern. Considering that the semantic interpretation procedure is based on probabilistic algorithms, this implies a remarkable accuracy of the Semantic Interpreter.

[1]The tokenization accuracy for local business test cases is based on the correct identification of an entity and related location-based concepts. As the local business discovery is based on an additional feature recognizing location-based entities that is performed after a semantic graph has been computed, there is no relation and focus pattern employed for the local business test cases.

	cor. Tokens	cor. Relations	cor. Focus
General Entities (EN)	0.925	0.8	0.8
General Entities (DE)	0.9	0.85	0.825
General Facts (EN)	0.8	0.75	0.75
General Facts (DE)	0.7125	0.675	0.675
Local Entities (EN)	0.975	1.000	1.000
Local Entities (DE)	0.925	0.95	0.95
Local Specified Parameters (EN)	0.9	1.000	1.000
Local Specified Parameters (DE)	0.925	1.000	1.000
Local Business (EN)	0.85	-	-
Local Business (DE)	0.975	-	-

Table 7.2: Average accuracies of the parser's components

7.3.2 System Scope

The system scope evaluation analyzed the quantity of retrieved search results as well as their relevance with respect to the underlying natural language query. For the scope analysis, the retrieved results of queries searching for local entities, local specified parameters and general entities have been analyzed. For evaluating the retrieved search results, 5 relevant test cases have been selected for each query type, processed by the system and compared to the results retrieved by the publicly available interfaces of the knowledge bases. All test queries have been performed with a maximum number of results of 50. The precision and recall values of the retrieved results were computed as described in section 7.2. The precision values of the retrieved results has been calculated as the number of relevant documents compared to the number of all results. For estimating the number of relevant search results that have not been found by the system, i.e. the recall of the system, manual queries have been performed by accessing the publicly available interfaces of the system's knowledge sources. The recall value has been computed as the number of relevant documents retrieved by the system, compared to the number of documents retrieved by the manual search with the publicly available user interfaces of the knowledge sources. Fig. 7.2 shows the average precision and recall values of the retrieved results, the average harmonic mean considering both precision and recall of the retrieved results is listed in Table 7.3.

All results show a high recall value, which indicates that data modules were able to retrieve a majority of the relevant documents available in the knowledge bases. As the recall values tend to be higher than the precision in all request classes, the results indicate that database queries have rather been formulated too general than too narrow. This conclusion is especially applicable to data modules

Type	HM
General Entities (EN)	0.85
General Entities (DE)	0.84
Local Entities (EN)	0.90
Local Entities (DE)	0.85
Local Specified Parameters (EN)	1.00
Local Specified Parameters (DE)	0.96

Table 7.3: Average harmonic mean of the system's search results

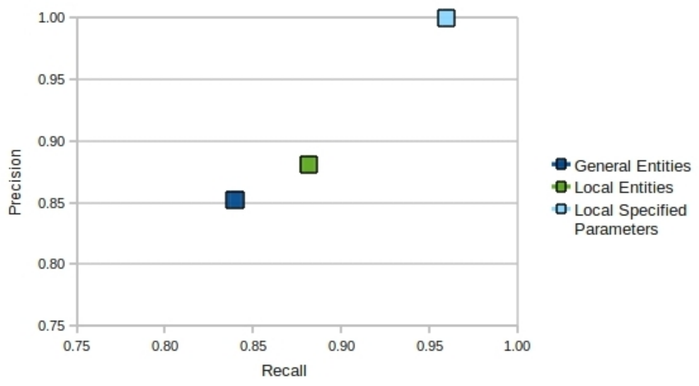


Figure 7.2: Average precision and recall of system results

accessing SPARQL endpoints, where queries including relations to resources have generally been modeled with non-terminal relationships. The high precision and recall values for queries focusing on customized event searches indicate that the concept of graph-based parsing including base token recognition is an effective concept for generating precise database queries.

7.3.3 Processing Time

For the purpose of identifying the efforts of processing client requests by the system, the processing time of the overall system as well as each component has been measured. The processing time evaluation was structured into the speed measurement of the tokenization and semantic interpretation, knowledge retrieval and the processing of the retrieved results. Furthermore, the processing time of each vocabulary and data module has been recorded. The average processing time required by the system for the query types presented in section 7.2 is depicted in Figure 7.3.

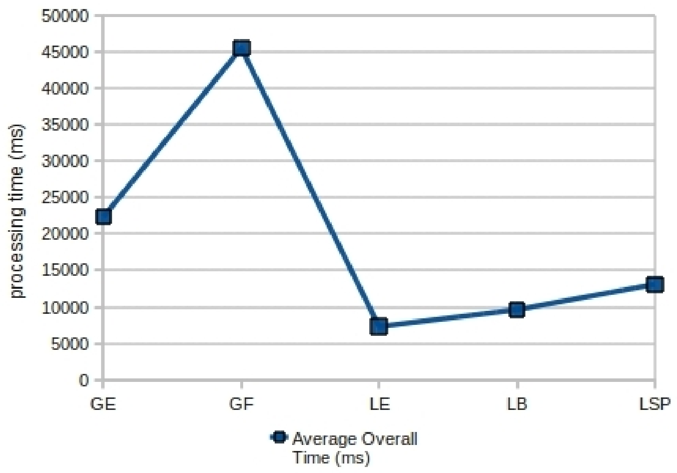


Figure 7.3: Average processing times of different query types

The test results show that queries for general knowledge have tended to require more processing time, with a significantly high processing time for queries searching for general facts. A remarkably fast processing speed has been reached by the test case searching for local entities. Especially local business queries have been computed significantly faster than other request types.

Tokenization	Semantic Interpretation	Query Processing	Result Processing
92.02%	0.065%	7.73%	0.12%

Table 7.4: Average share of the system's components of the overall processing time in percent

The average processing time required by each system component for the prototype's query types are displayed in Figure 7.4, displaying the average processing times of the Tokenizer, the Semantic Interpreter, the Query Performer and the Result Processor. The system speed evaluation revealed that a significantly high fraction of the measured processing time is accounted for the tokenization as well as the knowledge retrieval procedure. Since both components access data from external data sources and are accessed via web services, the processing times may

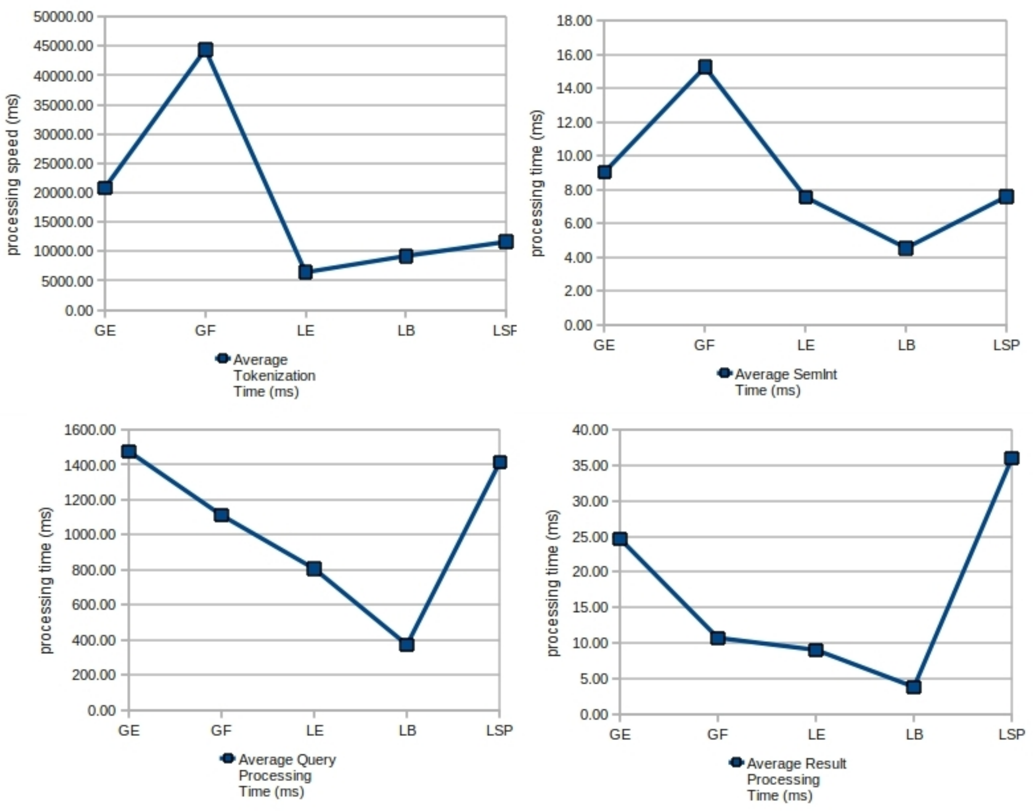


Figure 7.4: Average processing times of the systems components of different query types

partially be explained by the time required for data transmission. However, with a share of 92.02% the greatest share of the processing time is accounted for the tokenization procedure, indicating that the time required for database queries is a crucial factor for the system speed. Since the tokenization time required for computing queries for general facts has reached the highest value, it is likely that the time required by the Tokenizer correlates with the complexity of the natural language question. The small time intervals required by the semantic interpretation and result processing procedures thus indicate that the time required for internal data processing is relatively less significant. While queries for events with specified parameters have been parsed within an area comparable to local entity and local business questions, their required knowledge retrieval and result processing times were significantly higher.

7.4 Discussion

The evaluation has shown promising results for the model and architecture of a natural language search system presented in this thesis. The system is able to identify logical resources within an unstructured natural language query and translate it into a logical intermediate representation, which is then processed into a variety of knowledge base queries. The system is able to process precise location-based questions as well as general-knowledge queries and questions for specific facts about particular resources. Additionally, the prototypical implementation enables the concept of identifying and employing specific parameters contained in natural language within a customized event search. The retrieved results are merged, ranked and displayed in a human-readable form as defined in chapter 3. The system accuracy has shown a high consistency for all question types of the system. The proposed combination between shallow parsing techniques and a 3-staged graph-based model has fulfilled the requirements defined in chapter 3 and performed remarkably well. Room for improvement lies in the speed of the segmentation procedure of the tokenization algorithm, which has reached high values for complex natural language queries.

Outlook Finally, we will briefly discuss some approaches for further development of the system. The results retrieved by the speed measurement revealed how a significant portion of the processing time required by the system is accounted for the tokenization process. To improve the system speed, one would thus have to focus on the optimization of the database query process. As the prototype is based on web services for internal module interaction, the system could further be accelerated by switching to faster technologies. An important result of the speed evaluation is the observation that the majority of the required processing time is taken by the tokenization procedure, which even exceeds the processing time of the knowledge retrieval process on an average. Attempts to accelerate the query processing time thus have to aim at the time required by the vocabulary modules to query the system's databases in order to logically interpret natural language. As the Tokenizer computes all possible tokenizations for all possible term sets, the computing time increases largely with the length of the natural language input sentence. One possibility to simplify the tokenization is to discard unlikely term sets at an early stage. For example, the parser could start with the term set with the highest probability and accept a tokenization as soon as it exceeds a certain threshold. As the process of identifying tokens is one of the most computation-intensive tasks, the reduction of tokenization computations could increase the system's efficiency significantly. On the other hand, to discard possible parses carries the risk of discarding the relevant parses for a query.

Disambiguation Patterns The accuracy evaluation has shown a remarkable consistency of the Semantic Interpreter: If a query is structured correctly into semantic tokens, it is likely that the system also is able to compute the correct corresponding semantic graph. The highest potential to further improve the system accuracy is thus to extend the functionality for token disambiguation, i.e. the correct association of words with logical entities. One way to enrich the tokenization procedure would be the involvement of the query's context in the token disambiguation process, resulting in disambiguation patterns for token assignment employed analogously to node, relation and focus patterns employed by the Semantic Interpreter. Another issue that has arisen especially with the DBpedia ontology is redundancy within the terminological data corpus. For example, for identifying a populated place the DBpedia ontology knows the class dbpedia-owl:City as well as dbpedia-owl:PopulatedPlace. To improve the recall of the system's search results, it could be efficient to retrieve related classes or resources of query entities and to include them into the search process.

Lexicalization Considering the scope of the generated knowledge base queries, the queries generated for location-based requests have proved to be quite fitting. In case of queries generated for requests concerning general entities, the evaluation showed that the queries tend to be rather too general. Patterns consist so far of non-terminal symbols referring to token types such as classes, properties and instances. The approach of unlexicalized patterns, however, implies that terminal branches can only be generated if properties are included within the natural language query. For example, a query including only classes and instances such as *"cities in spain"* could only be parsed with a non-terminal relationship between the focus variable *?x* and the instance *"Spain"*. While the natural language query implies the search for cities geographically located in Spain, the generated SPARQL query would search for all cities with an undefined relationship to the country of Spain, which may result in unrelevant search results. A possible solution to this issue may be the employment of lexicalized graph patterns, i.e. the involvement of terminal symbols in patterns' expressions and annotations. This step would correspond to the existing approach of lexicalized PCFG introduced in section 2.2, which have achieved significantly high accuracy results and therefore be a promising outlook for further development of the system. On the other hand, as noticed in section 2.2 the development of lexicalized rules is a comprehensive, time-consuming task.

Multiple pattern employment The system's prototype employs so far only one relation pattern per token sequence parse. However, in case of complex queries it would be useful to enable the employment of several relation patterns to increase the probability to identify all node relations of the graph correctly.

User Dialogs An approach frequently realized by various works to resolve disambiguations in natural language parsing is the employment of user dialogs, which enables the users to choose the correct interpretation among a list of alternatives. Since this solution provides feedback about the correct interpretation of natural language queries, it is particularly interesting combined with machine learning techniques in order to improve the parsing accuracy over time. On the other hand, users may be interested in a fast and uncomplicated system and rather be unwilling to undergo an additional step until they receive search results by the system.

Machine Learning A promising extension of the system would be the employment of machine learning techniques on the semantic interpretation procedure. As each pattern is equipped with a truth value, an efficient mechanism to adapt the patterns' values after each parse could improve the system over time. Beside the correct parsing of natural language, machine learning techniques could further be employed for evaluating the relevance of the search results and to improve the ranking of the Result Processor. Another issue for optimizing the list of retrieved search results is the ranking improvement for general entities. An attempt to emphasize important search results would be to identify certain properties which may be consulted for instance rankings: For example, for a list of cities the property *population* could be a factor for ranking computation, resulting in a list emphasizing the size of a city in the computed search result list.

Chapter 8

Conclusion

This work presented the foundations of a distributed, ontology-driven natural language search system based on multiple, possibly heterogeneous data sources. We developed a parsing model consisting of a tokenization and a semantic interpretation generating a graph-based representation of a natural language query using probabilistic patterns. The approach uses an ontology-driven data model, structuring entities contained in the query in terms of classes, properties, instances and query constraints. The logical representation generated in the process of semantic interpretation depicts a natural language query in terms of a directed labeled graph, where nodes represent entities connected by edges indicating the entities' relationships. The semantic interpretation consists of three stages, constructing a semantic graph by 1) the generation of the graph nodes, 2) the generation of the graph relations and 3) the identification of the graph focus.

We developed a framework for an architecture of a distributed search system that is both independent with respect to the included vocabularies for parsing the syntax and semantics of a natural language query, as well as with respect to the employed knowledge sources, which deliver the base for retrieving search results. This functionality is achieved by modularizing the system, addressing external data sources by flexible modules that are specified to their corresponding data source. The main system communicates with the modules via well-defined interfaces and uses a triple-based data model to represent natural language parses. The system thus is able to parse a natural language query for multiple domains as well as retrieve search results from multiple data sources. Finally, the retrieved results are merged and ranked and presented enriched with additional meta data to the client.

The system evaluation consists of the testing of the system accuracy, the scope of the retrieved search responses as well as a measurement of the processing time. For the purpose of automatically asking natural language questions to the system, a set of test cases of different question types has been generated, in particular queries focusing on location-based resources (entities, businesses, events) as well as general resources and facts. The analysis of the accuracy test results have shown a high correctness of the syntactic parser. The scope evaluation revealed significantly high values for precision and recall of the retrieved search results, with the highest values achieved by location-based queries including specified parameters. This result indicates that the approach of token recognition and pattern-based semantic interpretation performs remarkably well and is able to improve the precision and recall of a search engine.

Appendix A

Document Type Definitions (DTD)s of System Interfaces

A.1 DTD of Vocabulary Module Responses

```
<!ELEMENT resultset (result*)>
<!ELEMENT result (id,tokentype,value,relevance,term,latitude?,
   longitude?,geoterm*,geotoken*)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT tokentype (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT relevance (#PCDATA)>
<!ELEMENT term (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
<!ELEMENT geoterm (#PCDATA)>
<!ELEMENT geotoken (#PCDATA)>
```

Listing A.1: Token response of a Vocabulary Module

A.2 DTD of Data Module Responses

```
<!ELEMENT resultset (result*)>
<!ELEMENT result (id,meta)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT meta (title,description?,uri?,image*,latitude?,
   longitude?)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
```

```
<!ELEMENT image (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
```

Listing A.2: Search response of a Data Module

A.2.1 DTD of System Responses

```
<!ELEMENT resultset (result*)>
<!ELEMENT result (id,relevance,category?,meta)>

<!ELEMENT id (#PCDATA)>
<!ELEMENT relevance (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT meta (title,description?,uri?,image*,latitude?,
   longitude?)>

<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT uri (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT latitude (#PCDATA)>
<!ELEMENT longitude (#PCDATA)>
```

Listing A.3: Search response of the system

Appendix B

Configuration

B.1 Vocabulary Module Configuration

```
uri=[uri]
path=[path]
[name=[vm_identifier]]
```

Listing B.1: Vocabulary Module configuration file (*config.txt*)

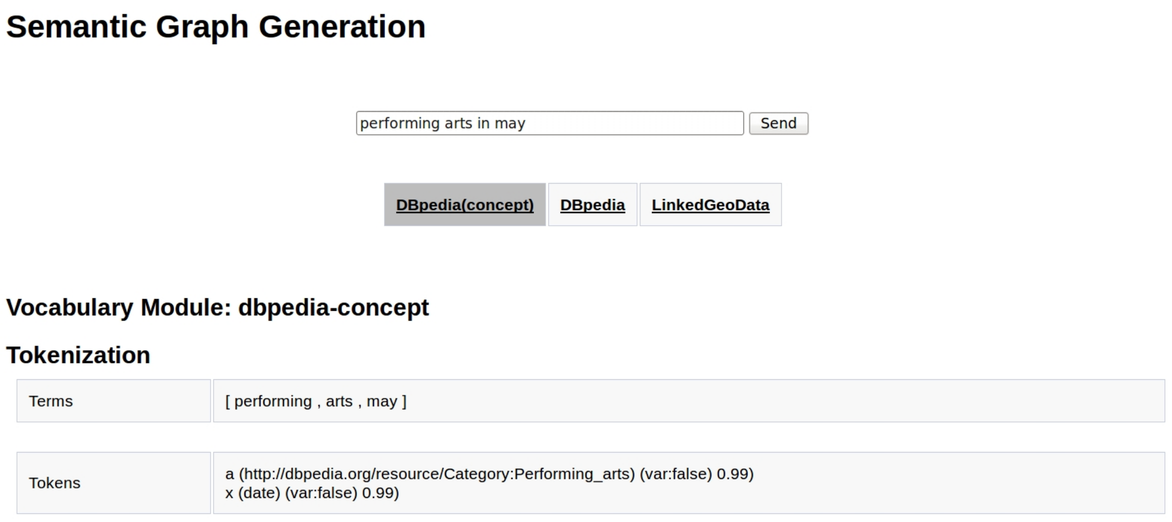
B.2 Data Module Configuration

```
base-uri=[uri]
path=[path]
vocabulary=[vocabulary_id]
[name=[dm_identifier]]
[logo=[logo_url]]
[app-key=[app_key]]
[focus-type=[focus_type_uri_0] ... [focus_type_uri_n]]
```

Listing B.2: Data Module configuration file (*config.txt*)

Appendix C

Semantic Graph Computation



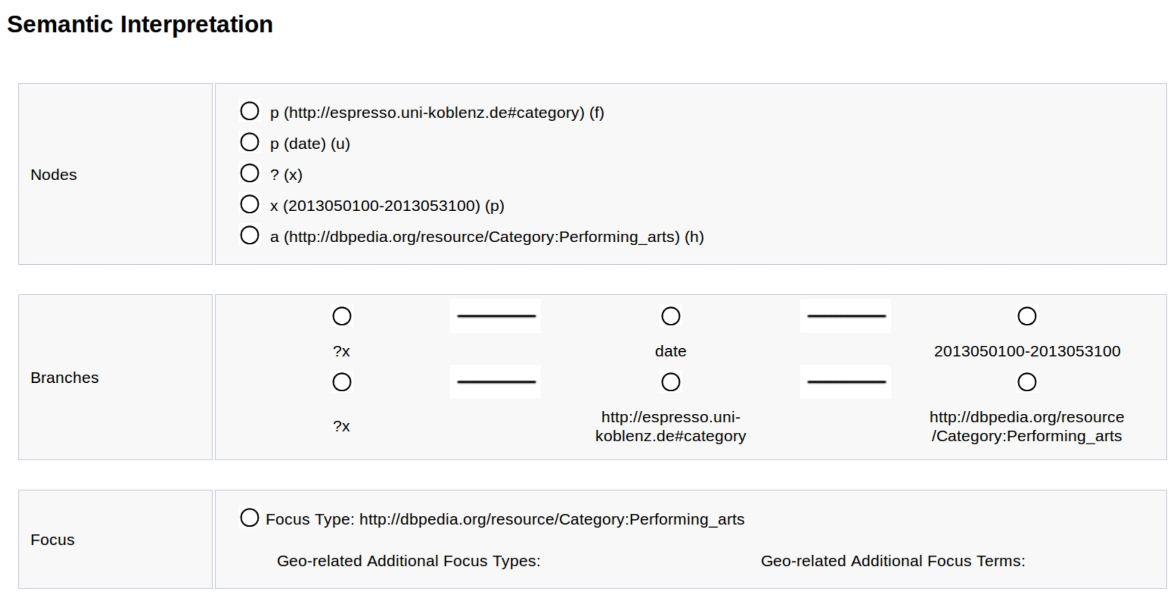
Semantic Graph Generation

Vocabulary Module: dbpedia-concept

Tokenization

Terms	[performing , arts , may]
Tokens	a (http://dbpedia.org/resource/Category:Performing_arts) (var:false) 0.99) x (date) (var:false) 0.99)

Figure C.1: Query specification comprising of a text form and the vocabulary selection as well as the tokenization of the query



Semantic Interpretation

Nodes	○ p (http://espresso.uni-koblenz.de#category) (f) ○ p (date) (u) ○ ? (x) ○ x (2013050100-2013053100) (p) ○ a (http://dbpedia.org/resource/Category:Performing_arts) (h)
Branches	○ ?x —— ○ date —— ○ 2013050100-2013053100 ○ ?x —— ○ http://espresso.uni-koblenz.de#category —— ○ http://dbpedia.org/resource/Category:Performing_arts
Focus	○ Focus Type: http://dbpedia.org/resource/Category:Performing_arts Geo-related Additional Focus Types: Geo-related Additional Focus Terms:

Figure C.2: Semantic interpretation of the query

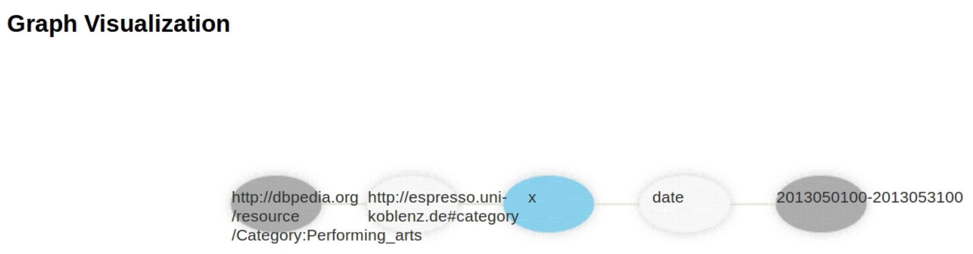
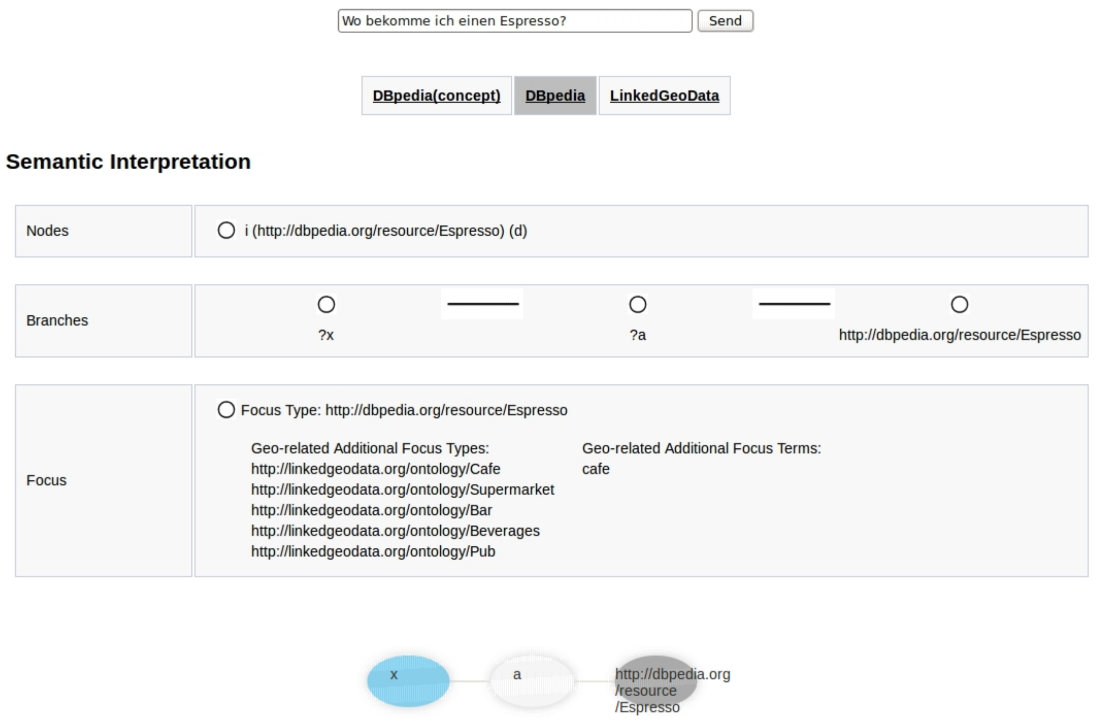

Graph Visualization
http://dbpedia.org/resource/Category:Performing_arts
http://espresso.uni-koblenz.de#category
x
date
2013050100-2013053100

Figure C.3: Visualization of the semantic graph realized by employing the JavaScript InfoVis Toolkit[a]

[a]http://philogb.github.io/jit, last visited on: May 31th, 2013.


Wo bekomme ich einen Espresso?

Semantic Interpretation

Nodes	○ i (http://dbpedia.org/resource/Espresso) (d)
Branches	○ ?x — ○ ?a — ○ http://dbpedia.org/resource/Espresso
Focus	○ Focus Type: http://dbpedia.org/resource/Espresso Geo-related Additional Focus Types: http://linkedgeodata.org/ontology/Cafe http://linkedgeodata.org/ontology/Supermarket http://linkedgeodata.org/ontology/Bar http://linkedgeodata.org/ontology/Beverages http://linkedgeodata.org/ontology/Pub Geo-related Additional Focus Terms: cafe

Figure C.4: Retrieval of related location-based concepts for a query's focus type

Bibliography

[1] Wintner, S. (2010) Formal language theory. *The Handbook of Computational Linguistics and Natural Language Processing*, vol. 57, pp. 11–42, Wiley-Blackwell.

[2] Charniak, E. (1997) Statistical techniques for natural language parsing. *AI Magazine*, **18**, 33–44.

[3] Cunningham, H., Maynard, D., and Tablan, V. (1999) JAPE: a Java Annotation Patterns Engine. Research Memorandum CS–99–06, Department of Computer Science, University of Sheffield.

[4] Androutsopoulos, I., G.D., R., and P., T. (1995) Natural language interfaces to databases - an introduction. *Journal of Language Engineering*, **1**, 29–81.

[5] (2013) Navigationsempfänger Chipsatz für Personennavigation mit Anwendungen bei erhöhter Genauigkeit (NAPA), URL: `http://projekt-napa.de`, NAPA Development Team, data provided on: May 14th, 2013.

[6] Bizer, C., Heath, T., and Berners-Lee, T. (2009) Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*.

[7] Heath, T. and Bizer, C. (2011) *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edn.

[8] Zaihrayeu, I., Sun, L., Giunchiglia, F., Pan, W., Ju, Q., Chi, M., and Huang, X. (2007) From web directories to ontologies: Natural language processing challenges. Technical Report DIT-07-029, University of Trento.

[9] Kaufmann, E. (2007) *Talking to the semantic web - natural language query interfaces for casual end-users*. Ph.D. thesis, Universität Zürich.

[10] Hirschman, L. and Gaizauskas, R. J. (2001) Natural language question answering: the view from here. *Natural Language Engineering*, **7**, 275–300.

[11] Schaefer, U. (2007) *Integrating deep and shallow natural language processing components: representations and hybrid architectures.*. Ph.D. thesis, Saarland University.

[12] Nederhof, M.-J. and Satta, G. (2010) Theory of parsing. *The Handbook of Computational Linguistics and Natural Language Processing*, Wiley-Blackwell.

[13] Erk, K. and Priese, L. (2008) *Theoretische Informatik*. Springer-Verlag, 3rd edition edn.

[14] Sipser, M. (2006) *Introduction to the Theory of Computation*. Thomson Course Technology, 2nd edition edn.

[15] Schabes, Y., Abeillé, A., and Joshi, A. K. (1988) Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. *Proceedings of the 12th Conference on Computational Linguistics-Volume 2*, pp. 578–583, Association for Computational Linguistics.

[16] Collins, M. (2003) Head-driven statistical models for natural language parsing. *Computational Linguistics*, **29**, 589–637.

[17] Klein, D. and Manning, C. D. (2003) Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, Sapporo, Japan, pp. 423–430.

[18] Collins, M. (1997) Three generative, lexicalised models for statistical parsing. *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, Stroudsburg, PA, USA, pp. 16–23, EACL '97, Association for Computational Linguistics.

[19] Charniak, E. (1997) Statistical parsing with a context-free grammar and word statistics. *Proceedings of the National Conference on Artificial Intelligence*, pp. 598–603, JOHN WILEY & SONS LTD.

[20] Miller, S., Fox, H., Ramshaw, L. A., and Weischedel, R. M. (2000) A novel use of statistical parsing to extract information from text. *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, pp. 226–233, Association for Computational Linguistics.

[21] Li, X. and Roth, D. (2001) Exploring evidence for shallow parsing. *Proceedings of the 2001 workshop on Computational Natural Language Learning-Volume 7*, p. 6, Association for Computational Linguistics.

[22] Abney, S. (1997) Part-of-speech tagging and partial parsing. *Corpus-Based Methods in Language and Speech Processing*, pp. 118–136.

[23] Sang, E. F. T. K. and Buchholz, S. (2000) Introduction to the conll-2000 shared task: Chunking. *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pp. 127–132, Association for Computational Linguistics.

[24] Daelemans, W., Buchholz, S., and Veenstra, J. (1999) Memory-based shallow parsing. *Proceedings of the Conference on Computational Natural Language Learning*, vol. 99, pp. 53–60.

[25] Schmid, H. (1994) Probabilistic part-of-speech tagging using decision trees. *Proceedings of International Conference on New Methods in Language Processing*, September.

[26] Merialdo, B. (1994) Tagging English text with a probabilistic model. *Computational Linguistics*, **20**, 155–172.

[27] Brill, E. (1995) Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, **21**, 543–565.

[28] Greene, B. B. and Rubin, G. M. (1971) *Automatic Grammatical Tagging of English*. Department of Linguistics, Brown University.

[29] Klein, S. and Simmons, R. F. (1963) A computational approach to grammatical coding of english words. *Jounral of the ACM (JACM)*, **10**, 334–347.

[30] Hindle, D. (1989) Acquiring disambiguation rules from text. Hirschberg, J. (ed.), *Proceedings of the 27th Annual Meeting on Association for Computational Linguistics*, pp. 118–125, Association for Computational Linguistics.

[31] Brill, E. (1992) A simple rule-based part of speech tagger. *Proceedings of the Workshop on Speech and Natural Language*, pp. 112–116.

[32] Roche, E. and Schabes, Y. (1995) Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, **21**, 227–253.

[33] Cutting, D., Kupiec, J., Pedersen, J. O., and Sibun, P. (1992) A practical part-of-speech tagger. *Proceedings of the 3rd Conference on Applied Natural Language Processing*, pp. 133–140, Association for Computational Linguistics.

[34] Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996) Mbt: A memory-based part of speech tagger generator. *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27.

[35] Ratnaparkhi, A. (1996) A maximum entropy model for part-of-speech tagging. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, April 16, vol. 1, pp. 133–142.

[36] Christodoulopoulos, C., Goldwater, S., and Steedman, M. (2010) Two decades of unsupervised pos induction: how far have we come? *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Stroudsburg, PA, USA, pp. 575–584, EMNLP '10, Association for Computational Linguistics.

[37] Chandra, Y. (2006) *Natural Language Interfaces To Databases*. Master's thesis, University of North Texas.

[38] Abney, S. P. (1991) Parsing by chunks. *Principle-Based Parsing: Computation and Psycholinguistics*, pp. 257–278, Boston: Kluwer Academic Publishers.

[39] Kudo, T. and Matsumoto, Y. (2001) Chunking with support vector machines. *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*, pp. 1–8.

[40] Sang, E. F. T. K. (2000) Noun phrase recognition by system combination. *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, pp. 50–55, Association for Computational Linguistics.

[41] Ramshaw, L. A. and Marcus, M. P. (1995) Text chunking using transformation-based learning. *Proceedings of the Third ACL Workshop on Very Large Corpora*, pp. 82–94, Cambridge MA, USA.

[42] Buchholz, W. D. S. (1999) Cascaded grammatical relation assignment. *Proceedings of EMNLP/VLC-99*, pp. 239–246.

[43] Veenstra, J. and Buchholz, S. (1998) Fast np chunking using memory-based learning techniques. *Proceedings of BENELEARN'98*, pp. 71–78.

[44] Sang, E. F. T. K. and Meulder, F. D. (2003) Introduction to the CoNLL-2003 shared task: Language-Independent Named Entity Recognition. Daelemans, W. and Osborne, M. (eds.), *Proceedings of CoNLL-2003 and the 7th Conference on Natural Language Learning*, pp. 142–147.

[45] Borthwick, A. (1999) *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University.

[46] Mikheev, A., Moens, M., and Grover, C. (1999) Named entity recognition without gazetteers. *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pp. 1–8, Association for Computational Linguistics.

[47] Chieu, H. L. and Ng, H. T. (2002) Named entity recognition: A maximum entropy approach using global information. *Proceedings of the 19th International Conference on Computational Linguistics (COLING'02)*, Taipei, Taiwan.

[48] Zhou, G. and Su, J. (2002) Named entity recognition using an hmm-based chunk tagger. *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Morristown, NJ, USA, pp. 473–480, Association for Computational Linguistics.

[49] McCallum, A. and Li, W. (2003) Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003-Volume 4*, pp. 188–191.

[50] Zhou, G. and Su, J. (2002) Named entity recognition using an HMM-based chunk tagger. *Proc. 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002).*

[51] Kazama, J. and Torisawa, K. (2007) Exploiting wikipedia as external knowledge for named entity recognition. *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 698–707.

[52] Nadeau, D. and Sekine, S. (2007) A survey of named entity recognition and classification. *Linguisticae Investigationes*, **30**, 3–26, publisher: John Benjamins Publishing Company.

[53] Carreras, X. and Màrquez, L. (2004) Introduction to the conll-2004 shared task: Semantic role labeling. *HLT-NAACL 2004 Workshop: Eight Conference on Computational Natural Language Learning (CoNLL-2004)*, Boston, pp. 89–97.

[54] Gildea, D. and Jurafsky, D. (2002) Automatic labeling of semantic roles. *Computational Linguistics*, **28**, 245–288.

[55] Thompson, C. A., Levy, R., and Manning, C. D. (2003) A generative model for semantic role labeling. Lavrac, N., Gamberger, D., Todorovski, L., and Blockeel, H. (eds.), *ECML*, vol. 2837 of *Lecture Notes in Computer Science*, pp. 397–408, Springer.

[56] Pradhan, S. S., Hacioglu, K., Ward, W., Martin, J. H., and Jurafsky, D. (2003) Semantic role parsing: Adding semantic structure to unstructured text. *ICDM*, pp. 629–632, IEEE Computer Society.

[57] Furbach, U., Glöckner, I., and Pelzer, B. (2008) Loganswer - a deduction-based question answering system. *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR-08), Sydney, Australia.*

[58] Dong, T., Furbach, U., Glöckner, I., and Pelzer, B. (2011) A natural language question answering system as a participant in human q&a portals. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-2011), Barcelona, Spain, July 2011*, pp. 2430–2435.

[59] Popescu, A.-M., Etzioni, O., and Kautz, H. (2003) Towards a theory of natural language interfaces to databases. *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pp. 149–157, ACM.

[60] Minock, M. (2005) A phrasal approach to natural language interfaces over databases. Tech. rep., tR UMINF-05.09, University of Umea.

[61] Katz, B. (1988) Using english for indexing and retrieving. *RIAO '88 Conference on User-oriented Content-based Text and Image Handling, Cambridge, MA.*

[62] Li, Y., Yang, H., and Jagadish, H. V. (2005) Nalix: an interactive natural language interface for querying xml. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pp. 900–902, ACM.

[63] Li, Y., Yang, H., and Jagadish, H. V. (2006) Constructing a generic natural language interface for an xml database. *Advances in Database Technology-EDBT 2006*, vol. 3896 of *Lecture Notes in Computer Science*, pp. 737–754, Springer.

[64] Burke, R. D., Hammond, K. J., Kulyukin, V., Lytinen, S. L., Tomuro, N., and Schoenberg, S. (1997) Question answering from frequently-asked question files: Experiences with the faq finder system. *Tech. Rep. TR-97-05, Departmenet of Computer Science, University of Chicago.*

[65] Jijkoun, V. and de Rijke, M. (2005) Retrieving answers from frequently asked questions pages on the web. Herzog, O., Schek, H.-J., Fuhr, N., Chowdhury, A., and Teiken, W. (eds.), *CIKM*, pp. 76–83, ACM.

[66] Lassila, O., Swick, R. R., Wide, W., and Consortium, W. (1998) Resource description framework (rdf) model and syntax specification. URL: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222, last visited on: May 30th, 2013.

[67] Lopez, V. and Motta, E. (2007) Aqualog: An ontology-driven question-answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, **5**, 72–105.

[68] Wang, C., Xiong, M., Zhou, Q., and Yu, Y. (2007) Panto: A portable natural language interface to ontologies. *The Semantic Web: Research and Applications*, pp. 473–487, Springer.

[69] Pérez, J., Arenas, M., and Gutierrez, C. (2006) Semantics and complexity of sparql. *The Semantic Web-ISWC 2006*, pp. 30–43.

[70] Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002) Gate: an architecture for development of robust hlt applications. *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, USA.

[71] Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.-C., Gerber, D., and Cimiano, P. (2012) Template-based question answering over rdf data. *Proceedings of the 21st International Conference on World Wide Web*, Lyon, France, April 16-20, pp. 639–648.

[72] Mithun, S., Kosseim, L., and Haarslev, V. (2007) Resolving quantifier and number restriction to question owl ontologies. *Third International Conference on Semantics, Knowledge and Grid*, pp. 218–223.

[73] Cimiano, P. (2004) Orakel: A natural language interface to an f-logic knowledge base. *Natural Language Processing and Information Systems*, pp. 401–406, Springer.

[74] Kaufmann, E., Bernstein, A., and Fischer, L. (2006) Nlp-reduce - a "naive" but domain-independent natural language interface for querying ontologies. ESWC.

[75] Tablan, V., Damljanovic, D., and Bontcheva, K. (2008) A natural language query interface to structured information. *The Semantic Web: Research and Applications*, pp. 361–375, Springer.

[76] Damljanovic, D., Agatonovic, M., and Cunningham, H. (2010) Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. *The Semantic Web: Research and Applications*, pp. 106–120, Springer.

[77] Ran, A. and Lencevicius, R. (2007) Natural language query system for RDF repositories. *Proceedings of the Seventh International Symposium on Natural Language Processing (SNLP)*.

[78] Kaufmann, E., Bernstein, A., and Zumstein, R. (2006) Querix: A natural language interface to query ontologies based on clarification dialogs. *5th International Semantic Web Conference (ISWC 2006)*, pp. 980–981.

[79] Lopez, V., Motta, E., and Uren, V. S. (2006) Poweraqua: Fishing the semantic web. *The Semantic Web: Research and Applications*, pp. 393–410, Springer.

[80] Katz, B., Felshin, S., Yuret, D., Ibrahim, A., Lin, J. J., Marton, G., McFarland, A. J., and Temelkuran, B. (2002) Omnibase: Uniform access to heterogeneous data for question answering. *Natural Language Processing and Information Systems*, pp. 230–234, Springer.

[81] Katz, B., Lin, J., and Felshin, S. (2001) Gathering knowledge for a question answering system from heterogeneous information sources. *Proceedings of the Workshop on Human Language Technology and Knowledge Management-Volume 2001*, p. 9, Association for Computational Linguistics.

[82] Grosz, B. J., Appelt, D. E., Martin, P. A., and Pereira, F. C. (1987) Team: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, **32**, 173–243.

[83] Kwok, C. C., Etzioni, O., and Weld, D. S. (2001) Scaling question answering to the web. *ACM Transactions on Information Systems (TOIS)*, **19**, 242–262.

[84] Schwitter, R. and Tilbrook, M. (2004) Controlled natural language meets the semantic web. *Proceedings of the Australasian Language Technology Workshop*, vol. 2, pp. 55–62.

[85] Bernstein, A., Kaufmann, E., and Kaiser, C. (2005) Querying the semantic web with ginseng: A guided input natural language search engine. *15th Workshop on Information Technologies and Systems, Las Vegas, NV*, pp. 112–126.

[86] Andrenucci, A. and Sneiders, E. (2005) Automated question answering: Review of the main approaches. *Third International Conference on Information Technology and Applications, 2005. ICITA 2005*, vol. 1, pp. 514–519, IEEE Computer Society.

[87] Noy, N. F. and Musen, M. A. (2000) Algorithm and tool for automated ontology merging and alignment. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Available as SMI Technical Repot SMI-2000-0831*.

[88] Maedche, A. and Staab, S. (2000) Discovering conceptual relations from text. *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, pp. 321–325, IOS Press.

[89] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999) The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab.

[90] Kleinberg, J. M. (1999) Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, **46**, 604–632.

[91] Sommerville, I. (2007) *Software Engineering*. Pearson Studium, 8th edn.

[92] Fielding, R. T. (2000) *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine, California.

[93] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2008) Dbpedia: A nucleus for a web of open data. *The Semantic Web*, pp. 722–735, Springer.

[94] Lehmann, J., Bizer, C., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009) DBpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, **7**, 154–165.

[95] Frank, A., Krieger, H.-U., Xu, F., Uszkoreit, H., Crysmann, B., Jörg, B., and Schäfer, U. (2007) Question answering from structured knowledge sources. *J. Applied Logic*, **5**, 20–48.

[96] Stadler, C., Lehmann, J., Höffner, K., and Auer, S. (2012) Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, **3**, 333–354.

[97] (2013) Navigationsempfänger Chipsatz für Personennavigation mit Anwendungen bei erhöhter Genauigkeit (NAPA), Project Website (2013). URL: `http://projekt-napa.de`, last visited on: May 19th, 2013.

[98] (2013) Navigationsempfänger Chipsatz für Personennavigation mit Anwendungen bei erhöhter Genauigkeit (NAPA), Local Project Website (2013). URL: `http://userpages.uni-koblenz.de/~napa/Web`, last visited on: May 31th, 2013.

[99] Resnik, P. and Lin, J. (2010) Evaluation of nlp systems. *The Handbook of Computational Linguistics and Natural Language Processing*, **57**, 271.

[100] Artstein, R. and Poesio, M. (2008) Inter-coder agreement for computational linguistics. *Computational Linguistics*, **34**, 555–596.

[101] Schmitt, I. (2005) *Ähnlichkeitssuche in Multimedia-Datenbanken - Retrieval, Suchalgorithmen und Anfragebehandlung*. Oldenbourg Wissenschaftsverlag.