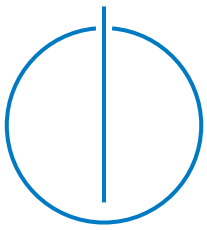
Department of Informatics
Technische Universität München

Master's Thesis in Information Systems

Semantic Analysis and Structuring of German Legal Documents using Named Entity Recognition and Disambiguation

Ingo Glaser



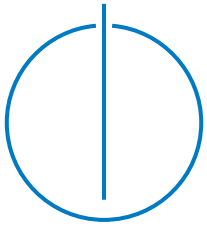
Department of Informatics
Technische Universität München

Master's Thesis in Information Systems

Semantische Analyse und Strukturierung von Deutschsprachigen Rechtstexten unter Verwendung von Named Entity Recognition Verfahren

Semantic Analysis and Structuring of German Legal Documents using Named Entity Recognition and Disambiguation

Author:	Ingo Glaser
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	M.Sc. Bernhard Waltl
Submission Date:	15.09.2017



I assure the single handed composition of this master's thesis only supported by declared resources.

Munich, 15.09.2017

Ingo Glaser

Acknowledgments

To begin with I would like to express my very great appreciation to my advisor Bernhard Waltl who supported me throughout my master thesis. I feel very fortunate to have worked with him in this area of research.

I would like to offer my special thanks to Elena Scepankova for her support.

I also want to thank Prof. Dr. Florian Matthes for his time, feedback and the opportunity to write this thesis at his chair Software Engineering for Business Information Systems (SEBIS).

Furthermore, I want to thank my three interview partners for their time and support: Dr. Stefan Blenk, Dr. Sebastian Bösing, and Marc Stolzki.

Most of all, I would like to thank my family and all my friends for their absolute support during my master thesis and the last years in general.

Zusammenfassung

Viele Industrien sehen die Digitalisierung immer noch als große Hürde an. Der juristische Sektor hat ebenfalls mit dieser Thematik zu kämpfen. Die Informationstechnologie erfährt im Hinblick auf diese Domäne eine zunehmende Bedeutung. Die wachsende Anzahl an digitalisierten juristischen Dokumenten, insbesondere an rechtliche Verträgen, untermauern diese These. Häufig liegen diese Dokumente jedoch lediglich in Form von unstrukturierten Daten vor und können so nur schwer von Computersystemen verarbeitet werden. Zudem enthalten diese Dokumente verschiedene Formulierungen für den gleichen Sachverhalt und überflüssige Informationen, die für den Leser nicht relevant sind. Dies erschwert die Nutzung von digitalen Verträgen zusätzlich. Trotzdem sind die semantischen Inhalte in diesen Dokumenten für den Leser von großer Bedeutung.

Diese Arbeit bietet eine Unterstützung für solche Geschäftsanforderungen durch eine Softwarekomponente, die es ermöglicht semantisch Analysen auf juritsichen Verträgen durchzuführen und diese strukturiert darzustellen. Um diesen Prozess umzusetzen werden Verfahren aus dem Natural Language Processing (NLP), wie beispielsweise Named Entity Recognition (NER) und Named Entity Disambiguation (NED), in eine Apache UIMA Pipeline integriert. Im Rahmen dieser Arbeit wird die existierende Funktionalität von Lexia, einer Platform für die kollaborative Bearbeitung und Analyse von juristischen Daten, genutzt. Dabei wird die entwickelte Softwarekomponente in Lexia integriert.

Ein neuer Ansatz für NER, welcher auf Verträge zugeschnitten ist und auf Vertragsvorlagen basiert, wird in dieser Arbeit implementiert. Dieses Vorgehen wird durch einen auf Templates basierenden NED Ansatz erweitert. Die Evaluation des entwickelten Systems zeigt die zuverlässige Anwendbarkeit dieses Vorgehens für juristische Verträge. Templatebasierte NER konnte einen F_1 von 0,92 vorweisen, während Implementierungen basierend auf GermaNER und DBpedia Spotlight lediglich einen Wert von 0,8 beziehungsweie 0,87 erreichen.

Abstract

Nowadays, many sectors face the obstacle called digitalization. So does the legal domain as well. The rising of legal technology is highlighted by the increasing number of digitized legal documents, in particular legal contracts. After capturing these, in many cases they are only available as unstructured data and thus barely processable by computer systems. However, the semantic knowledge within such a document is highly relevant to the reader. Furthermore, different contracts often incorporate diverse wording, while also including a lot of superfluous information. All these facts hamper the utilization of digitized legal contracts.

This work provides support for this business need by implementing a software component, enabling semantic analysis and structuring of legal contracts. In order to implement this process, common Natural Language Processing (NLP) tasks like Named Entity Recognition (NER) and Named Entity Disambiguation (NED) are incorporated into an Apache UIMA pipeline. In the course of this study, the existing functionality of Lexia a collaborative legal data science environment is utilized. Hereby, the software component being developed during this thesis is integrated into Lexia.

A new approach to NER, tailored to legal contracts, which are based on templates, called templated NER is implemented in the framework of this study. Then this method is enhanced by so called templated NED. The evaluation of the developed system, using German legal data, demonstrates the applicability of such approaches. Templated NER performed with an overall F_1 measure of 0.92, while implementations based on GermaNER and DBpedia Spotlight only achieved 0.8, respectively 0.87.

Keywords: Natural Language Processing, Named Entity Recognition, Named Entity Disambiguation, Named Entity Linking, Legal Text Analysis, UIMA, GermaNER, DBpedia Spotlight

Contents

Abbreviations

AI	Artificial Intelligence
AL	Active Learning
API	Application Programming Interface
BIO	Beginning-Inside-Outside
BIS	Business Information System
CAS	Common Analysis System
CRF	Conditional Random Field
DMP	Diff-Match-Patch
DT	Decision Tree
ECHR	European Court of Human Rights
FN	False Negatives
FP	False Posititives
FR	Function Requirement
HMM	Hidden Markov Model
HTML	Hyptertext Markup Language
HTTP	Hypertext Transfer Protocol
ICF	Inverse Candidate Frequency
IDF	Inverse Document Frequency
IE	Information Extraction
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
IS	Information System
IT	Information Technology
JSON	JavaScript Object Notation

MEM ..... Maximum Entropy Model

ML ..... Machine Learning

MLT ..... More Like This

NE ..... Named Entities

NED ..... Named Entity Disambiguation

NEL ..... Named Entity Linking

NER ..... Named Entity Recognition

NERC ..... Named Entity Recognition and Classification

NFR ..... Non-functional Requirements

NLP ..... Natural Language Processing

OPAC ..... Online Public Access Catalogue

PDF ..... Portable Document Format

POS ..... Part-of-speech

regex ..... Regular Expressions

REST ..... Representational State Transfer

RUTA ..... Rule-based Text Annotation

SML ..... Supervised Machine Learning

SOAP ..... Simple Object Access Protocol

SSML ..... Semi-supervised Machine Learning

SVM ..... Support Vector Machine

TF ..... Term Frequency

TF-IDF ..... Term Frequency-Inverse Document Frequency

TN ..... True Negatives

TP ..... True Positives

TV ..... Television

UI ..... User Interface

UIMA ..... Unstructured Information Management Architecture

URI ..... Uniform Resource Identifier

USML ..... Unsupervised Machine Learning

VSM ..... Vector Space Model

WSD ..... Word Sense Disambiguation

XML ..... eXtensible Markup Language

List of Figures

List of Tables

List of Listings

1 Introduction & Motivation

1.1 Motivation

The legal domain as well as the media domain both heavily rely on text as a central medium. 10 years ago, the media industry was one of the first industry sectors that was confronted with a rapid digitalization of business models, processes and the treatment of texts. For example, mediums like classic newspapers or television (TV) news were dominant in the consumption of information by humans. Nowadays, a large fraction of this information is consumed through internet offerings. Social media platforms such as Facebook[1], Twitter[2] or Instagram[3] has arrived from the bottom and disrupted the whole media industry. The disruption of existing behaviours is rather a typical feature of the digitalization[88].

We assume that the legal domain can expect a similar transformation in the next years. Trends and many start ups in the UK and in particular the USA already show, the legal domain is facing such a revolution as well. Due to the digitalization information can be found and consumed faster[131]. The rising of legal technology is highlighted by the increasing digitalization of legal documents as well[118].

When talking about digitalization, it must be distinguished between unstructured, semi-structured and structured data. In terms of digitizing texts these distinctions must be considered as well[66]. Section 1.2.1 explains the differences between these concepts. Structured data can be processed very well and it is the simplest way to manage information. However, semi-structured and

[1] `http://www.facebook.com`
[2] `http://www.twitter.com`
[3] `http://www.instagram.com`

in particular unstructured data is hard to process. Hence, transforming unstructured or semi-structured data into structured data is an important task in order to manage and process information[128].

Contracts are already digitized quite a lot. This is also due to the drafting process. Often a legal contract is not written with pencil and paper, but already created digital. Usually this results in semi-structured data. A huge added value can be created, when modeling and structuring these digitized legal documents properly[136]. This is in particular also true, due to the fact that lawyers and legal experts use different wording a lot. Having two lawyers creating two contracts with the same intent, the result is most likely two different contracts. Furthermore, legal contracts include a lot of information which is not highly relevant to the reader[66]. When there is a structured way of revealing the crucial information while neglecting superfluous passages of text of such a document, the resulting view would be the same. This is the main motivation behind this study.

The technical capabilities in order to accomplish such a task have arisen most recently. Intensive digital work is becoming more and more attractive, due to the increasing possibilities of text mining capabilities, support for data, time, and knowledge[136]. Not just the computational power increases yet continuously, but also new technologies such as Apache Spark[4] or Hadoop[5] originate and allow even more powerful clusters. Moreover, most recent research focuses a lot on *Natural Language Processing* (NLP) approaches. A lot of NLP techniques apply *Machine Learning* (ML) algorithms[30].

Having the legal technology on a growing branch, along with all the new technical capabilities, as well as the fact, that the structuring of text through computer-supported-analysis is very attractive for the legal domain, further research in this particular field is an interesting and promising task.

[4] `http://spark.apache.org`
[5] `http://hadoop.apache.org`

1.2 Basic Knowledge

In this section some fundamental terms and concepts for this work are explained. The described knowledge constitutes the basis for the rest of this thesis and thus is required for the following chapters.

1.2.1 Structured Information versus Unstructured Information

Information can be decomposed into structured, semi-structure, and unstructured information. Structured information refers to information having a predefined data model which allows machines to process this information as its interpretation is possible without ambiguities[52, 53]. Examples for representations of structured information are databases with a well-defined scheme or objects in object-oriented programming languages. Semi-structured information occurs for instance if the data is stored in the *eXtensible Markup Language* (XML) format. In contrast to this, unstructured information does not have a predefined data model and therefore the computer-aided processing is difficult, as the interpretation may be ambiguous[52, 109]. Examples for unstructured information are arbitrary texts, pictures or a scanned *Portable Document Format* (PDF) file. Unstructured information is the lion's share and fastest growing kind of the available information and thus, may contain lots of useful information, e.g. for companies[53].

1.2.2 Natural Language Processing (NLP)

Natural language processing is a research discipline focusing on analyzing unstructured information provided in natural language by the utilization of computer systems. For that reason, this unstructured information is enhanced with meta-information provided in a structured form. Natural language refers to human languages like English or German, for example. It may be provided in different forms like written text or spoken text[5, 82]. However, for the purpose of this work, only written text as input is considered. NLP is a pretty high-level discipline that can be decomposed into several sub-disciplines

like *Information Retrieval* (IR), *Information Extraction* (IE), machine translation or language generation, for instance. In this work the focus is set on the two former ones. Today, NLP is mainly an application of ML approaches, even though it still includes traditional methodologies such as rule-based techniques. More precise, it applies those methods to natural language. This is why NLP is also sometimes referred as computational linguistics[22].

NLP has received lots of attention during the last years due to the constantly improving performance of computer systems and increasing amounts of publicly available language resources like dictionaries and thesauri[75]. This evolution has been entailing new possibilities that have not been imaginable in the past. Distributed and concurrent processing is such a possibility that has been enabled by machines having several processors instead of only one. This has made it feasible to apply complex algorithms to large text collections, for instance.

1.2.3 Named Entity Recognition (NER)

NER is a task of NLP. Each continuous text contains proper names. Those may be names of persons, organizations or locations. *Named Entities* (NE) are phrases that contain these names[117].

Listing 1.1: Example of a setence containing *named entities*

```
The [ORG Sinc GmbH] located in [LOC Wiesbaden] is represented by its CEO Mr. [PER
    Martin Rollinger].
```

Listing 1.1 shows a sentence containing named entities. This sentence contains three named entities: (1) *Sinc GmbH* is an organization, (2) *Wiesbaden* is a location, and (3) *Martin Rollinger* is a person. "NER is an important task of IE systems."[117] The software component to develop in the context of this study is an IE system as well. Hence, NER is a key task of this work.

Chapter 2.2.1 discusses NER in more detail, before Chapter 2.3.1 handles about NER in particular within the legal domain. Eventually, Chapter 4.1 introduces some different approaches to conduct NER tasks.

1.2.4 Word Sense Disambiguation (WSD)

Another task of NLP is WSD. Depending on the application, WSD is closesly related to NER. Each natural language consists of thousands of different words. Nevertheless, concepts such as synonyms or acronyms exist. Due to this, many words have the same meaning as well as they have different meanings or senses[84]. There is an ambiguity about how to interpret a word, for such a word, having different meanings but given out of context[77]. As a simple example of ambiguity, consider the word *right* and two of the senses that can be found in the Oxford dictionary.

Listing 1.2: Example of the ambiguity of one word

– Morally good, justified , or acceptable.
– On, towards, or relating to the side of a human body or of a thing which is to the east when the person or thing is facing north.

Listing 1.2 depicts two of the meanings the word *right* has. Even though the two picked senses concern just one *Part-of-speech* (POS), that is adjectives, the two meanings diverge a lot. The first sense is rather ethical and deals with the question whether an action is morally accepted or not. On the other side, the second meaning is just about a direction. The task of disambiguation is to determine which of the meanings of a polysemous word is invoked in a particular use of the word[84].

There is also another kind of ambiguity, where a word can be used as different POS[84]. Going back to the previous example, *right* may be used as a noun, or as an adverb.

Listing 1.3: Example of the POS ambiguity of one word

– The human right to freedom of expressions applies to everyone.
– Please turn right on the next intersection .

Two sentences incorporating the word *right* are revealed in Listing 1.3. In the first example, *right* is clearly a noun, whereas in the second example, the word is used as an adverb. This determination of the usage of a word in terms of POS is called tagging[25]. These two different notions however clearly relate[73]. Using a word as an adverb instead of as a noun is obviously a different usage, with another meaning involved. Thus this can be seen as a

WSD problem[84]. Reversely, differentiating word senses may be viewed as a tagging problem, however using semantic tags rather than POS.

When we want to be able to link a NE towards a semantic function or tag, approaches of WSD are required. For the purpose of this work, only NEs need to be linked. The term used in this work, describing this linkage process is *Named Entity Disambiguation* (NED) and is introduced in Section 1.2.5.

1.2.5 Named Entity Disambiguation (NED)

The automatic understanding of the meaning of text has been a major goal of research in computational linguistics and is highly relevant for the industry as well[95]. NED heavily relates to NER. The aim of NED is to link NEs, discovered by NER, to an entry in arbitrary knowledge base. However, such an entry does not need to be necessarily in a knowledge base, it can be also part of a model or some other defined reference system[29, 39]. As long as the entry in the knowledge base provides some information gain, this task can be quite helpful in terms of understanding the semantic of a given text.

Chapter 2.2.2 treats NED in detail, before Chapter 2.3.2 is about NED within the legal domain. Later on in this thesis, Chapter 4.2 explains different approaches in order to conduct NED.

1.3 Structure

From an abstract view, this thesis compromises three major parts. First, a comprehensive literature review on subject-related topics is conducted. In the second part, based on the performed literature review, a software component for the semantic analysis and structuring of legal contracts is developed. Finally, the prototype is evaluated. Hereby, the work is structured as described below.

Chapter 1 provided already a short motivation along with an introduction. This also includes the explanation of some basic terms, highly relevant for this work. The next Chapter (2) deals with related work. The goal of this chapter is to familiarize the reader with previous approaches to NER and

NED. This is mandatory, because these approaches are incorporated heavily into this thesis.

Chapter 3 verbalizes the objectives of this work. For this purpose, the research questions of this study are defined. Furthermore, the research method this thesis is based on, is depicted. Now, the conceptual part of this thesis can be discussed. Chapter 4 first introduces different approaches to NER and NED. Each section contains a clear comparison of the different approaches. This work involves two existing systems, Lexia and SocioCortex. Those systems are explained in this chapter as well. Eventually, a requirement analysis is conducted in order to define a proper software architecture. The prototypical implementation is dealt with in Chapter 5.

A comprehensive evaluation of the developed prototype is conducted in Chapter 6. Finally, the results relevant to the research questions from all three parts of this work are discussed in Chapter 7.

2 Related Work

In this chapter two NLP tasks, which play an crucial role in this work, are described. Each task can be performed by utilizing various approaches. These approaches are described in Chapter 4. Specific frameworks, along with implementation details and ways how to incorporate them into an Apache *Unstructured Information Management Architecture* (UIMA) pipeline, are discussed in Chapter 5. Before discussing NER and NED, Section 2.1 deals with recent research in terms of structuring unstructured information. The second section of this Chapter relates to NER and NED in general, while the third section focuses on the legal domain.

2.1 From Unstructured Data to Structured Information

Business Information Systems (BIS) have been existing already for a long period. They have arisen from usual *Information Systems* (IS), which are socio-technical systems to cover information requests[65, 19]. A BIS is a group of interrelated components, that work collectively to carry on input, processing output and control actions in order to convert data into information. That information may be used to support forecasting, planning, control, coordination, decision making and operational activities in an organization[81]. When we look at the definitions of data and information, the task of such a BIS becomes pretty clear.

Data comes in form of a number, text or statement such as a measurement or a date, and hence is a raw fact. Information on the other side is processed data. The process of transforming data into information adds a sense to the data[56]. With other words, this process is the structuring of unstructured data. In the literature concerning this transformation, the term *from unstructured data to*

structured data or *from unstructured information to structured information* is used frequently, e.g. [139]. Those terms have one huge weakness, they jumble the definitions of data and information a lot, depending on the local distinctions. Thus a clear distinction for this study is necessary. The wording in this thesis conforms to the definitions discussed here.

Coming back to one of the key tasks of a BIS, the transformation of unstructured data into structured information. Blumberg already noticed, that "the management of unstructured data is recognized as one of the major unsolved problems in the *Information Technology* (IT) industry"[18]. This was true back then, when BIS were established. Nowadays, a lot more research has been done in the field of this transformation process. Most of the research belongs to the research of IR and IE.

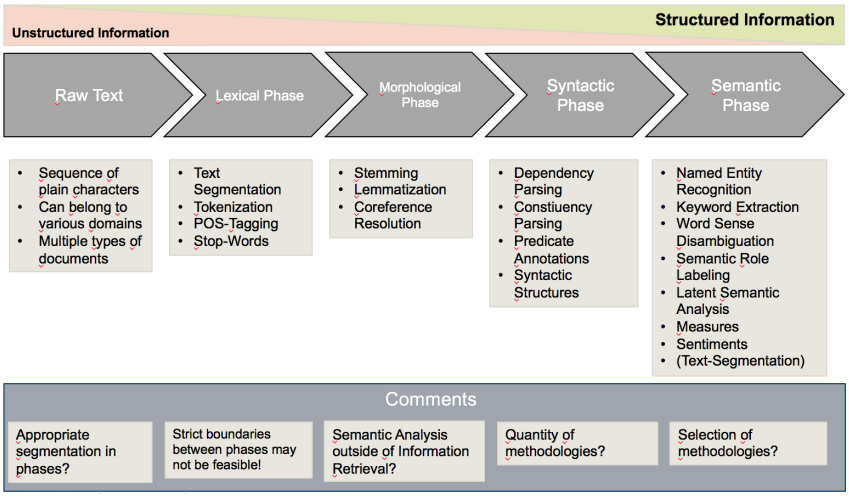


Figure 2.1: Text processing chain for IR

Source: own illustration based on [125]

Singh and Gupta came up with a text processing chain for IR[125]. Figure 2.1 outlines the key findings of Singh and Gupta. The information retrieval chain starts with *raw text*. This is a form of unstructured data, usually a sequence of plain characters. Such a sequence belongs to a specific document out of a corpus. Documents can belong to various domains, while multiple type of documents exist. While proceeding within the chain, data is structured during the first parts of the chain. The structured data can be used to extract semantic knowledge later on. Each method in the chain enriches the raw text with more structure. This is a continuous process whereas the transition to structured

information along with the semantic analysis is interlocking. Hence, there is no clear cut between the two phases[125].

The *lexical phase* converts a sequence of plain characters into a sequence of tokens. This is the first step towards structured information[62]. Approaches such as NER, POS tagging, stop-words, text segmentation or tokenization take place in that phase. NER is discussed in more detail in Section 2.2.1, Section 2.3.1 as well as in Chapter 4.1.

After the *lexial phase*, the *morphological phase* takes place. Linguistic morphology is the study of words[9]. This phase deals with parts of words, the structure of words, and with relationships between words.

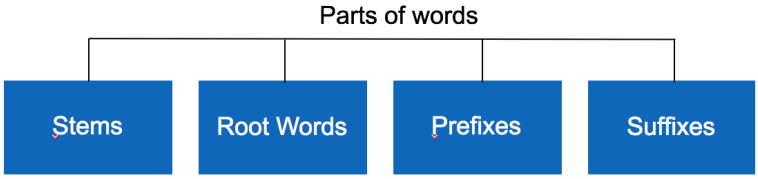


Figure 2.2: The different parts of words

Source: own illustration based on [9]

Figure 2.2 reveals the different parts of words. When looking at a word, there are mainly four parts of it[125]. The *stem* is common to all inflected variants of a word. Moreover, a word may have a *prefix* and a *suffix*. The *prefix* is an affix placed before the *stem* of the word and changes an existing word into another one. The *suffix* is very similar to a prefix, but the affix is placed after the *stem*. When a word has neither a *suffix*, nor an *prefix*, it is the primary lexial unit of a word and called a *root word*[21]. The literature suggest several methodologies concerning the morphological phase, including coreference resolution, lemmatization and stemming.

When the process has passed through the first two phases, the *raw text* has been already enriched with some structure. This is where the semantic analysis already takes part. It starts with the *syntactic phase*, attending to use syntactic characteristics in order to extract semantic knowledge[99]. Typical approaches of that phase are predicate annotations or syntactic structures.

The final phase of the processing chain for IR is the *semantic phase*. This phase eventually tries to understand the semantic of a given text[125]. In

other words, the structured and preprocessed text is used for some kind of onthology matching. The tagged entities are linked to semantic functions. Recent research and literature suggests a huge variety of methodologies. Those may be keyword extraction, latent semantic analysis, measures, semantic role labeling, sentiments, WSD or NED. Section 2.2.2 and Section 2.3.2 as well as Chapter 4.2 discusses NED in more detail.

The prototypical implementation being created during this thesis, incorporates just two phases of the here discussed process. During the lexical phase, in particular NER is performed. Moreoever, preprocessing tasks such as POS tagging or tokenization is used as well. Afterwards, the semantic phase is conducted in the form of NED.

2.2 NER & NED in General

In this section, NER and NED concepts from the literature are explained and discussed. However, this section deals with general approaches to NER and NED, loosened from the legal domain. In fact, the related work treated here is not binded to any domain. Section 2.3 refers to the legal domain in particular.

2.2.1 NER

Chapter 1.2.3 already introduced NER briefly. The first important task concerning this work is NER. Each continuous text contains proper names. The task of detecting and classifying those names is NER[12]. According to Surdeanu et al.[127], NER belongs to the methodologies of IE. Nevertheless, it does support and even accelerate IR a lot. This conforms to the text processing chain for IR[125]. While IE is more of NLP and ML to extract information from unstructured data, IR retrieves the information stored in some data storage.

There has been a lot of work on NER, in particular for the English language[117]. Jurafsky summarizes a lot of his research in his book Speech & Language processing[77, p. 349 ff.]. Borthwick[22] gives a good overview about the research in this field. When recognizing NEs, we distinguish between distinct

categories. The literature as well as various shared tasks suggest different categorizations. Borthwick[22] for instance, uses the following categories in his work: person, location, organization, date, time, percentage, monetary value, and "non-of-the-above". The CoNLL-2003[6] shared task suggests to use just three, respectively four, categories: person, location, organization, and other[117]. The same categorization has been used by Locke and Martin[83] for their NER classifier to extract NEs from tweets. Such a categorization of NE types often depend on the domain. For this work, the suggestion from CoNLL-2003 is assumed, enhanced by some of the categories from the literature. This leads to the following set of categories: person, organization, location, date, money value, reference, and other.

In the literature, different definitions and names are used to describe NER. Cardellino et al.[31] for instance, use the term *Named Entity Recognition and Classification* (NERC). However, here the classification does not refer to entity linking or WSD, but to the classification of NEs into the distinct categories. Collins and Singer[35] as well as Downey et al.[46] and Elsner et al.[49] split NER into two different tasks. They differentiate between NE segmentation and NE classification. In the first step, they segment and thus extract NEs, before they try to perform the classification into the different categories. Comparing to Cardellino[31], the segmentation complies to the recognition task, whereas the classification refers to the same in both approaches. Section 1.2.4 already introduced WSD. Later in this work (Section 2.2.2, Section 2.3.2, and Chapter 4.2), the relationship of WSD and NED is explained. However, nothing discussed in this section refers to either WSD or NED yet. In order to describe the task of recognizing NEs, the term NER is consistently used during this work. Hereby, NER is referred to the approaches described above.

Listing 2.1: Example of NEs including semantic information

```
Microsoft president Bill Gates.
```

Listing 2.1 gives a nice example to differentiate between NER and subsequent technologies. A NER system should report that *Bill Gates* is a person and *Microsoft* an organization. A further task would be to identify the relationship between *Bill Gates* and *Microsoft*. A further example shown in Listing 2.2 can

[6] http://www.conll.org

be used to describe the difference between the two tasks within NER and further steps, such as entity linking.

Listing 2.2: Example of a sentence to distinguish tasks of NER

```
Steve Jobs was the CEO of Apple.
```

The first task of NER, namely the NE segmentation, needs to identify *Steve Jobs* as well as *Apple* as NEs. It may be rather easy to categorize *Steve Jobs* as a person, nonetheless there is an ambiguity when talking about *Apple*. The classification task of NER must solve this ambiguity in order to tell that *Apple* is an organization in this case, rather than the fruit. Again, the relationship may be the task of a further component. Massive research has been done to solve issues like this one, but also to address other problems such as language portability[22]. Krupka and Hausman[79] developed a solution called netowl(tm) to address this issue. However, the majority of current NER systems support only single languages or they have different modes for some languages. GermaNER[7] developed by Benikova et al.[12] for instance, is a German NER tool. The Stanford University developed Stanford NER[8], a Java implementation of a NER, supporting different language modes like Spanish, German, English, and Chinese.

State of the art NER from the literature use a couple of different approaches for the actual implementation. Even though the majority of NER tools, use a combination of different technologies and features including Gazetteers, Mikheev et al.[92] developed a NER tool without a Gazetteer. They combine classical rule-based grammars with statistical models. The previously mentioned GermaNER uses a *Conditional Random Field* (CRF) classifier for their implementation. A pretty common technique used for NER is the *Maximum Entropy Model* (MEM). Systems by Bender et al.[10], Chieu and Ng[32] or Curran and Clark[40] use this technique. The results of the CoNLL-2013 shared task elucidate, that this model is a pretty good one for NER[117]. NER tools developed by Florian et al.[55], Klein et al.[78] or Mayfield et al.[87] used *Hidden Markov Models* (HMM), along with other approaches. Hammerton[63] even uses a neural network. All of the above mentioned systems are learning based. When talking about ML-based NER systems, we need to differentiate between unsu-

[7] `http://www.lt.informatik.tu-darmstadt.de/en/software/germaner`
[8] `http://nlp.stanford.edu/software/CRF-NER.shtml`

pervised, semi-supervised and supervised approaches. Obviously a variety of different techniques exist in each of these fields. According to Erik et al.[117], a combination of different learning systems has proven to be a good method for obtaining excellent results. Besides the ML approaches, the literature also suggests rule-based recognizer, as used by Mikheev et al.[92] and knowledge-based NER tools. DBPedia Spotlight[9] developed by Daiber et al.[41], which is discussed in more detail in Chapter 4.1.2.2, uses DBPedia[10] as a resource to identify NEs.

2.2.2 NED

During the introduction to this study, Chapter 1.2.5 already briefly introduced NED. It was defined as the process of linking a NE to an entry in some resource, which is the correct one for the context of occurrence. When we talk about linking or disambiguating NEs, the literature often uses the term *Named Entity Linking* (NEL) or NED for that task. In this thesis, the term NED is used in order to describe the task of linking a NE to a semantic function or role.

Manning dedicates a whole chapter in his book *Statistical NLP*[84, 229 ff.] to the linking of words to senses. He suggests different techniques for WSD: supervised disambiguation, unsupervised disambiguation as well as a dictionary-based disambiguation. The same suggestions are made by Jurafsky in his book *Speech & Language Processing*[77]. Obviously by applying small changes, those approaches may be feasible to NED as well. An approach to NED using Wikipedia as a knowledge base was implemented by Cucerzan[39], incorporating a *Vector Space Model* (VSM) for the disambiguation component. Nguyen and Cao[103] developed a hybrid system, using rule-based techniques in combination with a statistical approach. The novelty of their approach is that the disambiguation process is incremental and includes several rounds that filter candidates. Han and Zhao[64] developed a knowledge-based method to NED. Moro et al.[96] came up with an approach to NED, by merging the two worlds of WSD and NED.

All of the above mentioned approaches require some external source. This may be a knowledge base such as Wikipedia in order to perform the disambiguation,

[9] `http://github.com/dbpedia-spotlight/dbpedia-spotlight`
[10] `http://wiki.dbpedia.org`

but can be also an ontology. In fact, almost every external resource, adding semantic information to a NE, can be used for that purpose. This is necessary because the NEs need to be linked to something, in order to add value.

Listing 2.3: Example sentences to link NE to a knowledge base

– Michael Jordan is a researcher in Computer Science.
– Michael Jordan plays basketball in Chicago Bulls.
– Michael Jordan wins NBA MVP.
– Learning in Graphical Model: Michael Jordan.

Listing 2.3 gives four different example sentences. All four sentences include the NE *Michael Jordan*. After a NER has identified *Michael Jordan* as a NE and classified it as a person, the NED system needs to link the NE to a semantic role. A NED system should group the first and fourth *Michael Jordan* into one cluster for they both refer to the Berkeley professor *Michael Jordan*, meanwhile group the other two *Michael Jordan* into another cluster as they refer to another person, the Basketball Player *Michael Jordan*. Each cluster then links the NE to one entry in the knowledge base.

The task of NED may be an easy one for human beings. However, for a system this is a complex and difficult task[64]. This is the reason, why future research in this field is crucial[44].

2.3 NER & NED in Law

Approaches and techniques to NER and NED in form of recent work has been presented in the previous section (2.2). This section deals with related work in terms of NER and NED in the legal domain.

2.3.1 NER

In terms of NER, there is one key issue with portability, the language. Section 2.2.1 already expounded, that a NER system usually just works for one specific language. Nonetheless, the performance of a NER classifier can be amplified when developing it for a specific domain. This work deals with the semantic

analysis and structuring of German legal contracts. Hence, recent work in NER systems for the legal domain in Germany is crucial.

Developing a NER system for German is a difficult, but well researched task. German is a wide-spread and comparatively well-resourced language[12]. However, yet only three notable datasets exist, namely CoNNL-data[117], an extension to user-generated content by Faruqui and Padó[51] and the NoSta-D NE dataset[11].

Even though there has been a lot of German NE taggers, there is just one freely available developed by Benikova et al.[12]. Faruqui and Padó[51] created a German NER model for the Stanford NER, which is licensed under the GNU General Public License. Stanford NER is also known as a CRF classifier[54]. A NER system based on the MEM for German was developed by Bender et al.[10]. Chieu and Ng[32] as well as Curran and Clark[40] created similar systems for the German language in the course of the CoNNL-2003 shared task. Florian et al.[55] and Klein et al.[78] came up with an approach to German NER, using a combination of MEM and other techniques. The CoNNL-2013[11] shared task caused further research in German NER. ExB, also a system for NER based on German, was implemented by Hänig et al.[74]. This sytem won the CoNNL-2013 shared task.

A huge challenge for German NER tools is the fact, that not only proper nouns, but all nouns are capitalized. Due to this, the capitalization feature used in many NE classifiers is rendered less useful than in other Western-script languages, like English or Spanish[12]. On the other side, adjectives derived from NEs such as *english*, are not capitalized in German. This leads to a lower performance of NER for German than systems for English[11]. The good performance of English NER in comparison to German NER is also fostered by the fact, that most work in NLP has been done for English[51].

When dealing with the legal domain, the type system has to be extended. The categorization into persons, names and organizations, as done in general-purpose NER systems is not sufficient anymore. Names of laws, of typified procedures and even of concepts need to be regarded as well[31]. Going further, classifications of NEs may be differently as well. For instance, countries and organizations are classified as *Legal Persons*.

Listing 2.4: Example of a sentence including NEs in the legal domain

The [ORG Court] is not convinced by the reasoning of the [ORG combined divisions of the Court of Cassation], because it was not indicated in the [OTH judgment] that [LGPER Egitim−Sen] had carried out [OTH illegal activities] capable of undermining the unity of the [LGPER Republic of Turkey].

The example in Listing 2.4 illustrates this classification. In that example, *Egitim-Sen* as well as the *Republic of Turkey* are classified as *LegalPersons*. Even though legal informatics is on a growing limb[136], not much research has been conducted concerning NER in the legal domain. Dozier et al.[47] discusses NER in legal documents such as US case law, depositions, and pleadings and other trial documents. Hereby they differentiate between judges, attorneys, companies, jurisdictions, and courts as NE types. They outline three methods in their discussion: lookup, context rules, and statistical models. A nested NER system with neural networks was defined and implemented by Reimers et al.[112]. The system was developed during the GermEval-2014 shared task and got inspired by the findings of Collobert et al.[36]. As mentioned, there is not much more research in terms of NER within the legal domain, though there has been quite some research about IR and IE for the legal domain. Wyner et al.[140] have tried to extract arguments from legal cases, by developing a context-free grammar that allows the expressions of rules to identify those expressions. A powerful framework, based on Apache UIMA, to classify and annotate legal texts based on linguistic and semantic features was developed by Grabmeier et al.[60]. In his dissertation, Walter[135] focused on the extraction of legal definitions from the federal constitutional court only.

2.3.2 NED

While different languages constitute the major issue with NER systems, another portability problem relates in particular to NED. When developing a NED system, domain specific knowledge is required. As already discussed in Section 2.2.2, ML approaches are the state of the art for NED implementations. Such a system obviously needs to be trained. For that reason, a huge corpus of relevant (domain specific) training data is required. This is one of the major problems for NED in the legal domain, in particular for German, as not many datasets exist[11].

The literature reveals that most NED systems still use ontologies in order to enable the linking of NEs. Legal ontologies have been explored and developed for different purposed and even in various subdomains in recent research. Ajani et al.[1] came up with Syllabus, a legal taxonomy. Another legal core ontology LKIF[11] was created by Hoekstra et al.[71]. Breuker et al.[24] describe two legal ontologies in their work, FOLaw by Valente[130], and their own development, LRI-Core. Cardellino et al.[31] implemented a low-cost, high-coverage system for NER and NED for legal texts. Thereby, they incorporated the legal LKIF ontology as well as the YAGO[12] ontology. The LegalRuleML ontology [7] aims to represent machine-readable legal knowledge, with a particular attention to legal sources, time, defeasibility, and deonic operators. Furthermore a lot of general purpose ontologies with some legal content exist[31].

[11] http://www.estrellaproject.org/lkif-core
[12] http://www.yago-knowledge.org

3 Research Method

3.1 Research Questions

The objective of this study is the development of a prototypical environment in order to show the semantic content of contracts in a structured way. This environment is implemented within Lexia, a legal data science environment developed at the chair "Software Engineering for Business Information Systems" of the TU München. The Lexia environment is described in detail in Chapter 4.3.1.

In order to achieve these objectives, the thesis orients itself by means of the following research questions:

1. Which information does a stakeholder want to extract from legal contracts?
2. What are the functional and non-functional requirements of a software for the analysis of legal contracts?
3. Which NLP technologies can be used, to extract the semantic meaning of a legal contract? How to combine these technologies into an Apache UIMA pipeline?
4. How does a prototypical implementation enabling the semantic analysis of legal contracts look like?
5. How can such a system be integrated into the workflow of potential stakeholders?

Chapter 7.1 reflects on these research questions and answers them accordingly.

3.2 Research Method

As already stated in Section 3.1, the major goal of this study is the development of a prototypical implementation of a component, enabling the analysis of the semantic meaning of contracts and structuring these accordingly. Hence, this thesis reports the building process of this software component. In order to be able to develop a suitable concept for that implementation, this work is mainly focused to the *design science* approach shown by Von Alan et. al[133]. They also describe the *behavioral science*, which complements the *design science* well. The goal of *behavioral science* is the development and justification of theories, explaining or predicting issues related to identified business needs. On the other side, *design science* aims to develop and evaluate artifacts created to meed the before mentioned needs. Figure 3.1 reveals a conceptual overview, that combines *design science* and *behavioral science* paradigms to understand and evaluate IS research.

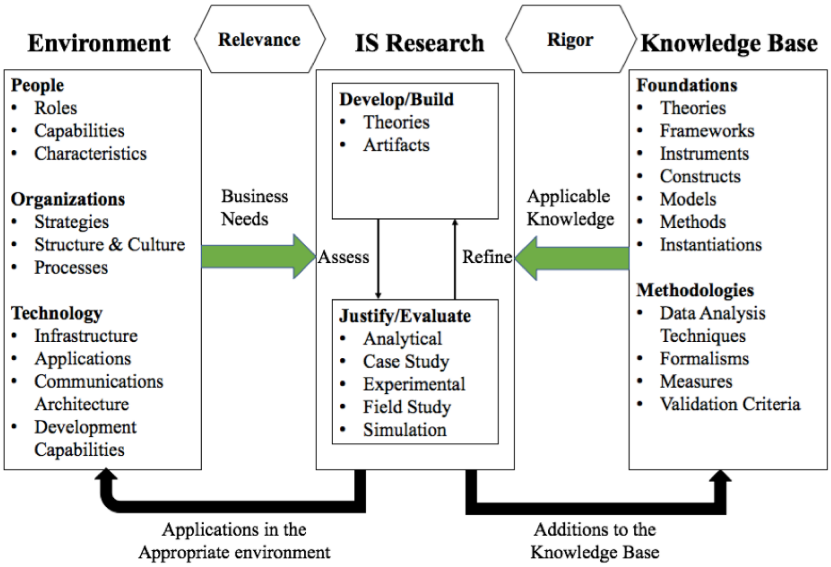


Figure 3.1: Information Systems Research Framework

Source: [97, p. 21] based on [133]

Even though in terms of this work, no research was conducted to determine business needs for such an application, during the literature review, those needs were identified. In order to improve this situation, this study focuses mostly on the design artifact. Thus, a prototypical implementation of a software component, to structure legal contracts properly is developed and evaluated.

The implementation orientates itself on the methodologies, frameworks and theories discussed during the literature review. The implemented prototype is evaluated by means of specific evaluation measures. The results obtained principally provide additions to the knowledge base, but may eventually also be applied in an appropriate environment to address the business needs identified.

To attain the basic knowledge required to develop the appropriate concept for the software component, as a first step a comprehensive literature study was performed [138]. The results of the literature study have been the following:

- Develop an overview of NER, its several forms and processes, as well as the terminologies used.
- Develop an overview of NED, its several forms and processes, as well as the terminologies used.
- Learn how NER and NED can be combined and integrated into an Apache UIMA pipeline.
- Discover and access NER and NED frameworks and systems that can be used for the semantic analysis of legal contracts.
- Discover and access visual frameworks that can be used to properly structure legal contracts.

In the course of this literature review, various sources such as conference papers, journals, books, blogs and reputable online pages were consulted. To access the relevant literature, online platforms such as the following were used:

- Google Scholar,
- Web of Science,
- Institute of Electrical and Electronics Engineers (IEEE) ,
- Online Public Access Catalogue (OPAC) ,
- and Google Books

Only primary literature were used during this literature review. The desired outputs of this literature review has been:

- an overview of related work,

- a set of requirements for such a software component, and
- a set of existing approaches to NER and NED

4 Concepts & Design

This chapters deals with crucial preparative steps for the prototypical implementation to semantically analysis and structure legal contracts. Section 4.1 discusses the different techniques to NER along with specific tools. Certain approaches to NED are explained in Section 4.2. These two sections also select freely available tools for the prototypical implementation. Since this implementation is based on already existing systems, those are explained in Section 4.3. To be able to develop the software component, requirements have to be elicited. This is done in Section 4.4. Based on the requirements analysis, a suitable software architecture is designed in Section 4.5.

4.1 Concepts of Named Entity Recogntion

4.1.1 Rule-based

The simplest approach to NER is based on rules. This is not surprising, as rule based approaches are still pre-dominant in many NLP tasks[33]. Rule-based NER approaches are broadly known as handcrafted approaches. This is because these systems are built by hand and rely heavily on the intuition of their human creators[22]. Such a system extracts the NEs by means of hard-coded rules. The rules are either defined within the source code of such an application or by means of a specific rule language, like Apache UIMA *Rule-based Text Annotation* (RUTA).

The New York University developed Proteus, a rule-based NER system[61]. It is implemented in Lisp and mainly composed of a large number of context-sensitive reduction rules. Another example has been implemented by Waltl et al.[136]. Based on a collaborative data science environment (see Section 4.3.1 for further information) and a large corpus from German tax law, they

demonstrate the extraction of semantic information. Their implementation uses Apache Ruta to express determined patterns. These pattern are used to extract the desired information. Even though this is no NER, even more complex structures are extracted. They were able to extract constitutions like legal definitions. Maat and Winkels[42] use *Regular Expressions* (regex) to extract sentences based on their occurring context within a law.

Listing 4.1: Example of rules for rule-based NER

```
- Title Capitalised_Word -> Title Person_name
  - Correct:
    - Mr. Jones
    - Gen. Schwarzkopf
  - Incorrect:
    - Mrs. Field's Cookies (A corporation)
    - Mr. Ten-Percent (nickname for a corrupt third-world official)

- Month_name number_less_than_32 -> Date
  - Correct:
    - February 28, July 15
  - Incorrect:
    - Long March 3 (a Chinese Rocket)
```

Some example rules from Proteus are shown in Listing 4.1. As the listing reveals, rule-based approaches entail drawbacks. In fact, for almost every NE rule there will be numerous exceptions. Given the usual time and resource constraints, it is almost impossible to code for every exception which one can think of, leaving aside exceptions which don't become apparent until one has run a test[22]. Furthermore the handcrafted approach implicates issues with the expense. It is pretty expensive to get a system up and running, requiring programmers with substantial experience in computational linguistics as well as domain knowledge. Thus, legal scientists are inalienable[136]. All this effort is obviously wasted, once the system shall be ported to either a new language or a new domain. Even when remaining in the same domain and language, adding new entity types to a document causes a rule-based NER system to fail[108].

4.1.2 Knowledge-based

The ecosystem of linked data develops[17] and so do the mutual rewards for structured and unstructured data providers alike. An increasing discoverability, reusability, and hence the utility of information can be achieved by higher interconnectivity between information sources[90]. By connecting unstructured information in text documents with linked data, records from the internet can be utilized, for instance to enhance information retrieval or to enable faceted document browsing[68, 91]. This is the reason for the development of knowledge bases and ontologies such as DBpedia or YAGO[13]. These public data infrastructures for a large, multilingual, semantic knowledge graph can be utilized to perform NER.

Systems applying such an approach differ from common NER tools quite a bit, depending on their complexity. There are complex implementation like Artequakt, which combines expertise and experience from several projects (for further information see Section 4.1.2.1), or DBpedia Spotlight (for further information please refer to Section 4.1.2.2). These systems incorporate typical NLP methodologies into the NER task. Tokenizer, lemmatizer, POS-tagger or other techniques are used for pre-processing, before a single token is requested against the knowledge base. The *Uniform Resource Identifier* (URI) of one-to-many resources is returned. Afterwards the token is linked to that URI, which corresponds to the recognition of a NE. A huge benefit of such an approach is the information attached to the knowledge base[90]. This is not just beneficial for eventual visualizations, but also for the case of ambiguity. In the event of a returned tuple of URIs, a system may perform a classification based on the context it knows, to select the proper resource. Other systems do not solve those ambiguities. A very simple and straight-forward approach to a knowledge-based NE recognizer would be a simple annotator. Given a textual representation, an annotator can sequentially process each word and match it with the knowledge base. By doing so, the majority of the NEs within that text are identified, while having the issue of extracting superfluous information.

The following two sections (4.1.2.1 and 4.1.2.2) introduce two examples of knowledge-based NER representatively.

[13] `http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago`

4.1.2.1 Artequakt

Artequakt is a project linking a knowledge extraction tool with an ontology to achieve continuous knowledge support and guide IE. It mainly tries to identify NEs along with their existing relationships using ontology relation declarations as well as lexical information. Hereby Artequakt combines experience and expertise of three projects: (1) Artiste[14], (2) The Equator IRC[15], and (3) The AKT IRC[16]. The project is developed for the artists and paintings domain[2].

In the first step, they created an ontology for that specific domain. Afterwards, a huge variety of IE tools has been used to populate that ontology with information extracts from online documents. The populated ontology is stored in a knowledge base, while the stored data is consolidated. The second step consisted of the development of narrative construction tools to query the knowledge base through an ontology server to search and retrieve relevant facts.

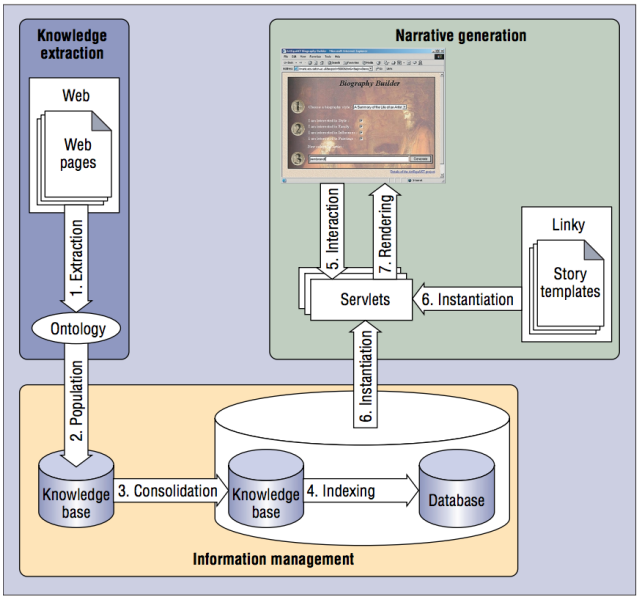


Figure 4.1: The Artequakt Architecture

Source: [2]

Artequakt's architecture unites three key areas, as shown in Figure 4.1. The knowledge extraction tools collect information items along with sentences and

[14] http://users.ecs.soton.ac.uk/km/projs/artiste
[15] http://www0.cs.ucl.ac.uk/research/equator
[16] http://www.iam.ecs.soton.ac.uk/projects/akt

paragraphs from web documents. These information fragments are passed to the ontology server along with metadata derived from the ontology. Second, the information is consolidated and stored by the ontology server. The Artequakt server takes user requests to generate the requested content via a web interface. Even though the artists and paintings domain is far away from the legal domain, this approach can be easily applied to arbitrary domains[2] and thus to the legal domain as well.

4.1.2.2 DBpedia Spotlight

DBpedia is an interlinking hub in the web of data, enabling access to many data sources in the linked open data cloud[90], licensed under the GNU license. DBpedia contains about 3.5 million resources from Wikipedia. The majority of the resources is classified into a consistent cross-domain ontology. The ontology is populated with classes such as places, persons or organizations. Furthermore, fine-grained classifications like soccer players or IT companies is existing. Resources possess attributes as well as relations to each other[41].

Mendes et al.[90] developed DBpedia Spotlight Annotator to enable the linkage of web documents with that hub. It is a system to perform annotation tasks on text fragments, such as documents, paragraphs or sentences, provided by a user. Hereby, the user wishes to identify URIs for resources mentioned within that text. This can be seen as a typical NER system.

The system is based on an approach with four different stages. The first stage recognizes the phrases in a sentence that may indicate a mention of a DBpedia resource, this stage is called spotting. In the next step candidates are selected, that may map the spotted phrase. In the most cases a bundle of resources is identified and hence, an ambiguity is current. This is when the disambiguation stage takes place. It tries to discern the context of the spotted phrase in order to decide on the best choice amongst the candidate resources. Eventually the annotation takes place. Within that stage, the user is able to inject configuration parameters to satisfy its specific needs[90].

Spotting

The spotting algorithm uses an extended set of labels in a lexicalization dataset to create a lexicon for spotting. This lexicalization set is created by utilizing the graph of labels, redirects and disambiguations in DBpedia. This lexicon associates multiple surface forms to a resource and interconnects multiple resources to an ambiguous label. The labels used for the lexicon are created from Wikipedia page titles. Textual references are extracted in the form of wikilinks within Wikipedia. The actual implementation for the spotting algorithm is based on the LingPipe Exact Dictionary-Based Chunker[4].

Candidate Selection

For the candidate selection, the DBpedia lexicalization dataset is used to determine candidate disambiguations for each surface form. This step is crucial, since it offers the possibility to narrow down the space of disambiguation possibilities. However, this step has a downside as well as a benefit. Obviously, narrowing down the candidates for the disambiguation improves the time performance. On the other side, performing a too aggressive candidate selection may reduce recall. For that reason, Mendes et al.[90] decided to apply minimal pre-filtering but introduce a post-disambiguation stage where the user can define his configuration for appropriate results[41].

Disambiguation

The DBpedia resource occurences are modeled in a VSM where each DBpedia resource is a point in a multidimensional space of words. In comparison to classical IR systems where the *Term Frequency* (TF) measures the relevance of a term in a document, TF represents the relevance of a word for a given resource in the model of DBpedia. Furthermore, as common to *Term Frequency-Inverse Document Frequency*(TF-IDF), the *Inverse Document Frequency* (IDF) weight depicts the general importance of the word in the whole DBpedia corpus. However, when dealing with the importance of a word for disambiguation, it turned out that IDF fails to capture adequately[41]. This is why the *Inverse Candidate Frequency* (ICF) was introduced. ICF is based on the explanation of Deng et al.[43]. The discriminative power of a word is inversely proportional

to the number of DBpedia resources it is associated with, which is the intuition behind ICF. In what follows, the mathematical definition of ICF is provided.

Let R_s be the set of candidate resources for a surface form s. Let $n(w_j)$ be the total number of resources in R_s that are associated with the word w_j. Then ICF is defined as:

$$ICF(w_j) = log(\frac{|R_s|}{n(w_j)}) = log(|R_s|) - log(n(w_j)) \tag{4.1}$$

To measure uncertainty in probability distributions, entropy has been commonly used[123]. With regard to a word's association with DBpedia resources, the entropy of a word can be defined as:

$$E(w) = -\sum_{i \in R_s} P(r_i|w) log(P(r_i|w)) \tag{4.2}$$

If that word w is connected to those resources with equal probability $P(r|w) = 1/n(w)$, the maximum entropy is transformed to $E(w) = log(n(w))$. The maximum entropy is used to approximate the exact entropy in the ICF formula. This is, since generally the entropy tends to be proportional to the frequency $n(w)$.

Having now that VSM representation of DBpedia resources with TF*ICF weights, the disambiguation task can be seen as a ranking problem with the objective to rank the correct DBpedia resource at the first position. The candidates resources are ranked according to the similarity score between their context vectors and the context surrounding the surface form. For the similarity measure, *cosine* is used[41].

Configuration and Annotation

DBpedia Spotlight uses a different approach in terms of configuration than most systems. While many of the current approaches for annotating content tune their parameters to a specific task, DBpedia generates a number of metrics to inform the users and let them decide on the policy that best suits their needs. This introduces more flexibility to various use cases in comparison to the classical approach[90]. The system offers five configuration parameters. First

the user can decide on a target set of resource types, based on the DBpedia ontology. In order to avoid the annotation of rare resources, the user can specify a resource prominence. Due to the fact that the disambiguation process returns a similarity score, it can be used to configure topic pertinence. Another configuration can be done in terms of contextual ambiguity. Eventually, the user can define a disambiguation confidence[90].

Usage of DBpedia Spotlight

The DBpedia Spotlight is available in two forms: a web application as well as a web service.

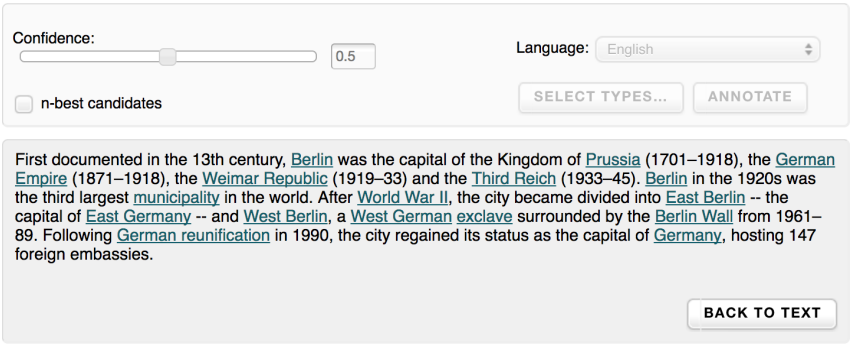


Figure 4.2: The UI of the DBpedia Spotlight Web Application

Source: Own screenshot

The UI of the web application is shown in Figure 4.2. The web application serves as a demo to test and visualize the results of different service functions. The user is able to enter text into a text box. On user's request, the system highlights the surface forms and creates associations with their corresponding DBpedia resources. Additionally, a separate disambiguation stage can be executed. In this case, the system bypasses the spotting phase and only annotates selected phrases[90].

The web service of DBpedia Spotlight facilitates the integration into external applications. In fact, two instances of the web service exist. There is a *RESTful* (Representation State Transfer) and a *Simple Object Access Protocol* (SOAP) web service. A user can access both, the annotation and disambiguation process and define all existing configuration parameters. The web service return different formats such as *Hypertext Markup Language* (HTML), XML

or *JavaScript Object Notation* (JSON)[41]. The web service of DBpedia is explained in Chapter 5.1.3.2 during the implementation phase of this work.

4.1.3 Machine Learning

As already discussed in previous chapters, the main task of a NER system is to recognize previously unknown NEs. An ability like this, hinges upon recognition and classification rules triggered by distinctive features associated with positive and negative examples[100]. As described in Section 4.1.1, early studies were mostly based on handcrafted rules. Recent work however focuses on ML techniques. The most recent NER approaches rely on *Supervised Machine Learning* (SML). By means of SML, such a system either automatically induces rule-based systems or a sequence labeling algorithm starts from a collection of training data. SML has one common issue that is, training data is necessary in order to train the classifiers. Even though a variety of automatic approaches to create training data exist, manual work is always necessary. Due to this, handcrafted rules remain the preferred technique when no sufficient training data is available[121]. The fact that five out of eight systems were rule based in the MUC-7 competition[100], but sixteen presented systems at CoNNL-2013 dedicated to ML, acknowledges this assumption.

In terms of ML, different methodologies exist. As already mentioned, SML is one of the possibilities. *Semi-supervised Machine Learning* (SSML) as well as *Unsupervised Machine Learning*(USML) complete the round of ML approaches. The following sections deal with these three approaches in more detail.

4.1.3.1 SML

The idea of SML is to study the features of positive and negative examples of NEs over a large training collection of annotated documents and design rules that capture instances of a given type in unannotated documents. Hereby different techniques can be used, such as HMM[16], *Decision Trees* (DT)[120], MEMs[23], *Support Vector Machines* (SVM)[6] or CRFs[89]. All these variants utilize the same concept. A large annotated corpus is analyzed to memorize a list of entities. This list is transformed into rules to detect such entities based

on a set of distinctive and discriminative features. These rules are then applied to new documents in order to extract similar entities.

When developing a ML-based NER system, an evaluation is crucial in order to be able to measure the performance of such a system. Furthermore, different systems should be comparable as well. The concept of baselines is used for this purpose. A baseline is a method that uses heuristics, simple summary statistics, randomness or ML to create predictions for a dataset. This prediction is used to measure the baseline's performance, such as accuracy, precision and recall. The resulting metric will become what any other system is compared against. A common baseline SML method consists of tagging words of a test corpus when they are annotated as entities in the training corpus[100].

The main intuition behind this work is to incorporate NER approaches in order to create a prototypical implementation for the semantic analysis and structuring of legal contracts. Hereby, also SML-based NER approaches shall be incorporated. However, details of all ML possibilities along with their detailed techniques are not the focus of this thesis. Hence, the above described techniques are not discussed in detail. Nevertheless, selected NER tools are elucidated along with the techniques used by those in the following section. For further information about the other techniques, please refer to the references above.

4.1.3.1.1 Approach to NER of Cardellino et al.

Cardellino et al.[31] developed a system to link NEs to a structured knowledge representation, the LKIF ontology[59]. Hereby, they linked the LKIF ontology to the YAGO ontology[126] and through it, taking advantage of mentions of the NEs in Wikipedia. Their system is divided into two tasks. First, they recognize NEs and classify them into a class. This is how NER is defined for this work. Second, they link the NEs to YAGO URIs, which is NED. This section describes their work in order to enable NER, while the discussion about their NED implementation takes place in Section 4.2.1.3. In order to do so, first they manually linked the LKIF ontology to the YAGO ontology, before training their classifiers for NER and NED.

LKIF is an abstract ontology describing a core of legal concepts. It consists of different modules with high-level concepts, and also three modules with

law-specific concepts[59]. The ontology constitutes a total of 69 law-specific classes. However, LKIF is not populated with real-world entities[31]. This is where YAGO captures an important role. It is a knowledge base extracted from Wikipedia, WordNet[17], and GeoNames[18]. Furthermore, it is linked to the DBpedia ontology and to the SUMO ontology[19]. Thereby it covers knowledge of more than 10 million entities, while containing about 120 million facts about these entities. According to Cardellino et al.[31], this information was evaluated to be above 95% accuracy.

The mapping between the two ontologies was carried out manually and as follows: for each LKIF concept, they try to find an equivalent in YAGO. If there is not such an equivalent, then they look first for a subclass, then for a superclass. When an equivalent concept has been found, they establish the alignment using the OWL primitives *equivalentClass* and *subClassOf*. Next they navigate YAGO to visit the related concepts and to check whether they could be aligned with other LKIF concepts. Since all children nodes of a connected node are connected by their most immediate parent, all children nodes of the aligned YAGO nodes are effectively conntected to LKIF. Hence, LKIF acts as the backbone of the newly created ontology, which can be thought of an LKIF extension, including the alignment of the concepts with YAGO ones[31]. They performed this matching not on the base of relations, but only classes. The process ended up in 30 classes from LKIF, which could be mapped to a YAGO node. From YAGO, 47 classes were mapped to LKIF, with a total of 358 classes considering child nodes. This summed up in a total of 4.5 million Wikipedia mentions. In order to create their corpus, Cardellino et al. extracted a dump of the Englisch Wikipedia. After several preprocessing steps, they extracted those sentences, that contained at least one mention of a named entity. In that process, a NE is a mention that has a link to an entity of YAGO, that belongs to the newly created ontology[31].

Now they were able to build different NER classifiers. Hereby they trained a SVM as well as the StanfordNER. StanfordNER could not handle the level of granularity of their ontology though. Furthermore, they learned a neural network with one hidden layer. This approach ended up in a NER classifier, for the legal domain in Englisch. The evaluation of their classifiers was done

[17] `http://wordnet.princeton.edu`
[18] `http://www.geonames.org`
[19] `http://www.adampease.org/OP`

by means of the test portion of the Wikipedia corpus, but also on a judgment corpus. Therefore Cardellino et al. manually annotated excerpts from 5 judgments of the *European Court of Human Rights*(ECHR). They achieved a F_1 of 0.24 for the SVM and 0.86 for the neural network over six NE classes on the Wikipedia corpus. When sticking to the legal domain, by evaluating on the judgment corpus, only a F_1 measure of 0.56 for StanfordNER and 0.47 for the neural network was reached. This reveals also the entitlement of this work, to ground a base for further research in the field of NER and law.

4.1.3.1.2 GermaNER

GermaNER is a generic German NE tagger that can be readily used from command line or integrated into any NLP application to automatically tag NEs. For the latter sense, the tagger is available as an Apache UIMA component. A crucial contribution to the NER community has been made, due to the fact that this system is under a permissive license that allows academic and commercial use without licensing fees. According to Benikova et al.[11], this is one of the sole freely available NER tools.

This system integrates a CRF[80] for sequence tagging. CRFs are scalable, highly accurate and easy to use as the training data can be prepared without the need of ML experts[70]. The *CRFsuite* by Okazaki[105] has been integrated into a *clearTK* UIMA framework[13]. This enables more convenient training, feature annotation, classification and entity extraction[12]. Hereby the system is highly configurable, as it allows the user to either use the built-in model, or train it with new training data and feature sets, while the standard model is optimized with the existing feature set. Furthermore GermaNER offers a technique of data chunking. This allows users with powerful machines to use larger data chunks, while users with weaker machines can still run the system with smaller data chunks[12].

A nice benefit of this system is its NER tagger pipeline. The pipeline consists of distinct components integrated into an UIMA[52] pipeline written in the Java programming language. This allows other developers and researchers to easily integrate the system into other NLP applications, for instance to use GermaNER as a reasonable baseline system to further expand training data

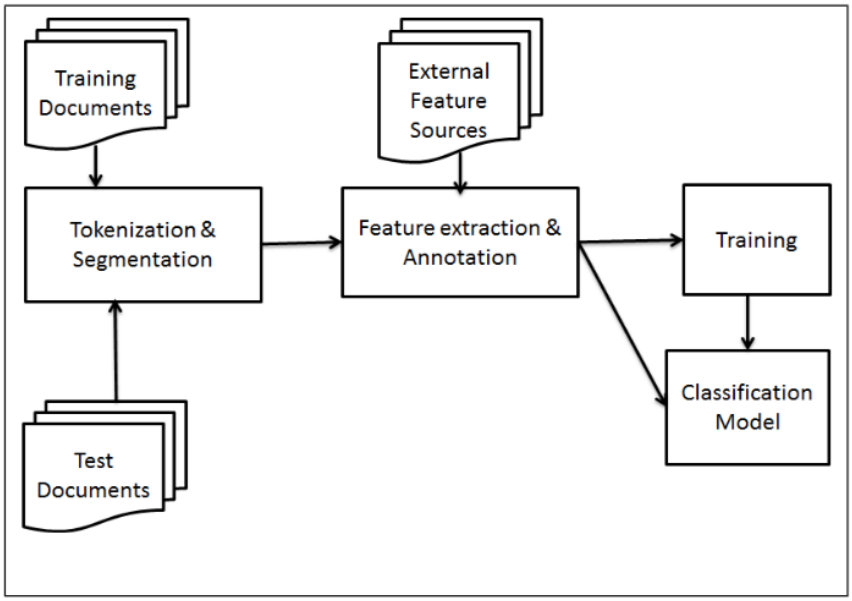


Figure 4.3: The tagger pipeline of GermaNER

Source: [12]

sets using *Active Learning* (AL) and adaptive annotation approaches[12]. Figure 4.3 illustrates this pipeline. The first component conducts tokenization and segmentation on the training and testing data. The results of this component are stored in a UIMA *Common Analysis System* (CAS) object. Such a CAS object contains the currently processed data as well as meta information[52]. Next, the feature extraction and annotation component follows. Different features are obtained by the feature extractors. Hereby the features are received either from the token and surrounding tokens, such as word and character n-grams, or the features are supplied from external sources, such as gazetteer lists or lists induced by unsupervised methods. In doing so, this component internally annotates the document accordingly. A further component is the training element. It produces a CRFsuite classifier model based on the annotated features. The final component is a classifier component where unseen documents, which get feature annotated in a similar way as the training data, are subject to prediction of NEs[12]. This NER tagger is designed in a way, that each mentioned component can be replaced or modified easily.

GermaNER only accepts the CoNLL-2013 format as input. Such a file contains one token per line, while sentence should be separated by a blank line. The output of the tagger is a tab separated file. The first column corresponds to the same as in the input file. In the second column, the predicted NE tag

is stored in form of the *Beginning-Inside-Outisde* (BIO) scheme. The BIO-scheme suggest to learn classifiers that identify the beginning, the inside and the outside of the text segments[111]. An exemplary output in that scheme is shown in Table 4.1.

Table 4.1: Exemplary output of GermaNER

Token	**Entity Type**
Dan	B-PER
Jurafsky	I-PER
is	O
a	O
leading	O
researcher	O
in	O
NLP	B-OTH
at	O
the	O
Stanford	B-ORG
University	I-ORG

In the development of a SML-based NER system, the creation and feature selection is crucial[12]. GermaNER uses a variety of features, whereby a configuration file is used to enable and disable various available features. In the remaining paragraphs of this section, the features, consolidated into groups, incorporated by GermaNER are presented.

Character and Word Features

The first feature group consists of the first and last character uni-, bi-, and trigrams of the current token. This may be for instance prefixes and suffixes, time-shifted from -2 to +2. Another influential feature for the system are character category pattern features, which are extracted from the current token based on unicode categories[20]. Moreover, the words themselves are used as features in a window between -2 and +2[12].

[20] `http://www.unicode.org/notes/tn36/`

NE Gazetteer

The gazetteer features used in GermaNER were created through the assembling of several lists containing NEs. Besides other gazetteers such as a personal name list extracted with the NameRec tool from the ASV toolbox by Biemann et al.[14] from large, publicly available corpora, several Freebase lists were used for this purpose. The foremost lists are merged into a *GazetteerFeature*, whereas the Freebase lists build an own feature *FreebaseList*. Freebase[20] is an English community curated database containing well known places, people and things under CC-BY license.

POS

GermaNER utilizes automatically induced POS tags as POS features. Clark[34] created a system which clusters words into different classes in an unsupervised manner, based on distributional and morphological information. The POS induction of GermaNER is based on Clark's system. 10 million sentences from the Leipzig Corpora Collection[21][114] were used to induce 256 different classes[12].

Word Similarity

The four most similar words of the current token, received from the JoBimText[22][15] distributional thesaurus database, made available in a window of size 2, form this feature group[12].

Topic Clusters

A fixed number of topic clusters, most of which are quite pure in terms of syntactic and semantic classes, is used. By applying LDA topic modeling[23] to above mentioned JoBimText German distributional thesaurus, using the thesaurus entries as documents for LDA, this topic clusters were created. This is similar to the approach in the ExB system by Hänig et al.[74]. Benikova

[21] `http://www.corpora.uni-leipzig.de`
[22] `http://www.jobimtext.org`
[23] `http://gibbslda.sourceforge.net`

et al. have generated different sets of these clusters, each for all words and for uppercase words only, and use the number of its most probable topic as a token's feature. This is time-shifted in a window range of -2 to +2[12].

Other Features

Two further features were implemented in GermaNER. The *position* feature reflects the position of the token in the sentence. A differentiation between uppercase and lowercase, the beginning of a sentence, camel case and all uppercase is made with the *case* feature, time shifted between -2 and +2[12].

4.1.3.2 SSML

The main concept behind SSML is called bootstrapping, involving a smaller degree of supervision, such as a set of seed records, for starting the learning process[100]. A example is a system targeted at law names, asking the user to provide a small number of example names. Now the system searches for sentences containing these names and attempts to identify some contextual clues common to the given example names. Then, the system tries to find other instances of law names that appear in similar contexts. The learning process is then reapplied to the newly found examples, so as to discover new relevant contexts. A large number of law names and a huge number of contexts will eventually be gathered, by repeating this process. Nadeau et al.[101] report performances that rival baseline SML approaches, in recent experiments in SML-based NER. As in SML, SSML needs this seed set in order to start the learning process. For this bootstrapping process, classic rule-based approaches can be used to create seed records[22].

Different techniques to implement a SSML-based NER system exist. Brin[26] utilizes regex in order to generate list of book titles paired with book authors. These regex implement the lexial features which are used for the generation of book titles paired with book authors. The technique starts with seed examples like *"Dan Jurafsky, Speech & Language Processing"* and uses fixed lexical control rules, like a regex to describe a title. Many web sites conform to a acceptable common format across the site. This fact it utilized in the approach of Brin. Once a given web site is found to contain seed examples, new pairs

can often be identified using simple constraints. Such an constraint may be the presence of an identical context around an interesting pair. As an example, having the passage *"Speech & Language Processing, written by Dan Jurafsky"* would enable finding, on the same web site, *"Foundations of Statistical Natural Language Processing, written by Christopher D. Manning"*.

Collins and Singer[35] parse a complete corpus in search of candidate NE patterns. Hereby, a proper name followed by a noun phrase in apposition may be a pattern. An example of a pattern like this may be *"Tim Cook, the CEO at Apple Inc."*. Such a pattern can be identified by using POS tagger. Patterns are kept in pairs *{spelling, context}* where *spelling* associates to the proper name and *context* refers to the noun phrase in its context. Again starting with an initial seed of spelling rules, the candidates are determined.

Listing 4.2: Seed example of spelling rules

```
Rule 1: if the spelling is San Francisco then it is a Location
Rule 2: if the spelling contains Mr. then it is a Person
Rule 3: if the spelling is all capitalized then it is a law name
```

An example of such a seed set of spelling rules is shown in Listing 4.2. Those candidates, satisfying a spelling rule are classified accordingly and their contexts are stored. A set of context rules is created by storing the most frequent contexts found. Repeating this process over and over, contextual rules can be used to find further spelling rules.

Mutual bootstrapping that consists of growing a set of entities and a set of contexts in turn, was introduced by Riloff and Joness[115]. Contrary to working with predefined candidate NEs, they start with a few seed entity examples of a given type. All patterns found around these seeds are accumulated in a large corpus. Then the found contexts are ranked and used to find new entities. A similar approach has been developed by Chucchiarelli and Velardi[38]. However, they use syntactic relations to discover more accurate contextual evidence around the entities. Furthermore, they waive the idea of human generated seeds, but incorporate existing NER systems for the creation of their seeds.

SSML-based NER approaches are able to reach performance measure similar to SML-based techniques, while starting with a seed of 10 examples and resulting in one million entities with a precision about 88%[107]. A key issue when

working with SSML and thus, with bootstrapping is the selection of unlabeled data[100]. Heng and Grishman[76] figured that the selection of documents using IR-like relevance measures and selection of specific contexts bring the best results.

A quite new approach to SSML is *Active Machine Larning* (AML). The intuition behind an AML procedure is to overcome the necessity of having a seed set. The training data is created in an interactive loop with the support of an *oracle*. This oracle may be a human expert, being able to identify NEs. The system asks queries in the form of unannotated instances to be labeled by the human annotator. The predictive model then can be trained by those labeled instances. While recurring this process, it is likelier that the classifier will perform well. This is because of the hypothesis behind AML, that is if the learning algorithm can select the data from which it learns, it will perform well with a small training set[122, p. 11 ff.].

This section already introduced some techniques to approach SSML-based NER. However, no specific NER system relevant for this work was found, which would be worthy to introduce.

4.1.3.3 USML

In general USML is a clustering task[100, 84]. Clustering is an unsupervised data-analytics technique to identify hidden patterns in data[69]. Of course this can be applied to NER as well. For example, a system can try to gather NEs from clustered groups based on the similarity of context. Other techniques exist also. Most of the approaches rely on lexial resources, on lexical patterns and on statistics computed on a large unannotated corpus. This is necessary because simple clustering of NEs indeed create groups, but the system cannot know which type a certain group belongs to.

The problem of labeling an input word with an appropriate NE type is studied by Alfonseca and Manandhar[3]. They take NE types from WordNet. The methodology is to assign a topic signature to each WordNet synset by merely listing words that frequently co-occur with it in a large corpus[100]. Now, the word context of a given word in a given document is compared to type signatures and classified under the most similar one.

Hearst[67] described the method to identify hyponyms and hypernyms. This method is applied by Evans[50] in order to recognize potential hypernyms of sequences of capitalized words appearing in a document.

NEs often occur synchronously in several news articles, whereas common nouns do not[124]. Shinyama and Sekine also noticed a strong correlation between being a NE and appearing punctually and simultaneously in multiple news sources. With that in mind, identifying rare NEs in an unsupervised manner also in combination with other NER techniques can be useful and was applied by them.

Even though some research in USML-based NER systems has been done, yet these approaches do not reach baseline performances as SML or SSML does[77]. As a consequence, no USML-based NER approach is implemented or utilized in this work.

4.1.4 Templated

In the course of this study, a new approach to NER in contracts is developed. This approach is called *templated NER*. The creation of a contract is mainly a manual and intensive task. Legal practitioners need to be able to understand requirements of a deal to define a suitable contract[116]. However, often existing contracts are refined, instead of creating a new contract from scratch. Over time, this lead to the existence of contract templates. For simple circumstances such as a rental deal, contract templates exist. The legal expert only needs to fill the placeholders with the respective information. Contract creation via templates is pretty common today[93]. Having this in mind, NER can be achieved easily on contracts, defined by a template, as long as the template is at hand as well.

The intuition behind this templated NER approach is that, if we compare an actual contract with its template, only the populated information remains as differences. When thinking about relevant information in a contract, mostly NEs emerge. With other words, when a contract template is filled in the majority of information are NEs. Of course this method basically just picks off the low hanging fruits, nonetheless it is a valid NER system for that specific kind of contracts.

The implementation of such a technique is expounded in Chapter 5.1.3.3, during the implementation phase. Moreover, the disambiguation of NEs towards a semantic model can be achieved with such an approach as well. The idea behind this is discussed in Chapter 4.2.3.

4.1.5 Comparison

The previous section discussed the different approaches to NER. This section compares the different techniques by means of different criteria. Each criteria is assessed with either *low*, *medium* or *high*, whereby the assertion depends on the actual criteria. Moreover, *none* can be used if the criteria is not applicable.

Table 4.2: Comparison of NER approaches

Criteria	**RB**	**KB**	**SML**	**SSML**	**USML**	**T**
Language Portability	Low	High	Medium	Medium	High	High
Domain Portability	Low	Medium	Medium	Medium	High	Low
Required Training Data	None	None	High	Medium	Low	None
Expert Knowledge	High	Medium	Low	Low	Low	Low
Manual Work	High	Medium	Medium	Low	Low	Low
System Complexity	Low	Medium	Medium	High	High	Low

Table 4.2 gives an overview about the comparison of the different approaches to NER. The newly introduced abbreviations for this table are as follows: *RB* is refers to rule-based, knowledge-based is referred to *KB*, and *T* means templated. A rule-based system seems to be not too complex, while it can be created without training data. However the downside of such an approach is its rigidity. The templated approach is very portable in terms of language, because a template just needs to be translated. On the other side, domains

cannot be changed that easy. Looking at the table above, templated NER seems to be a perfect fit. Nonetheless, it must be remarked, that this approach only works when templates are available, while having the limitations of the NEs marked as such in the template. Knowledge-based approaches look quite promising as well. They have decent portability capabilities, while requiring some expert knowledge in combination with manual work. As long as no ML is incorporated, training data is not necessary either. Speaking about ML-based approaches, USML seems to be a good-looking option, as long as the fact is neglected, that it is very hard to perform. Yet SML-based approaches are superior, but the trend may switch over to SSML.

For this study, three different concepts of NER shall be implemented. Having three main classes of NER approaches, ML-based, knowledge-based and templated, barring rule-based approaches, one technique out of each class is used. Due to the fact that research is highly advanced in rule-based NER systems, along with the fact that the chair of "Software Engineering for Business Information Systems" of the Technische Univeristät München already performed quite some research in this field, such an implementation is avoided. Two freely available system are selected, GermaNER is used to represent a SML-based NER system and DBpedia Spotlight for the sake of knowledge-based tools. Obviously the templated NER approach is implemented in the course of this thesis and thus also integrated into the prototypical implementation.

4.2 Concepts of Disambiguation

The major goal of this study is to semantically analyze and structure legal contracts. For this purpose only NEs shall be used. Recognized entities are subject to be linked to types and attributes of a semantic model, describing a contract. Section 4.5 as well as Chapter 5.1.4 discuss this approach in detail. This disambiguation process can be achieved by utilizing NED. WSD was already introduced in Chapter 1.2.4. It was defined as the process of linking a word to a sense, which is the correct one for the context of occurrence. NED and WSD both address the lexical ambiguity of natural language. However while the two tasks are pretty similar, they differ in a fundamental aspect. In NED the textual mention can be linked to a NE which may or may not contain

the exact mention, while in WSD there is a perfect match between the word and the sense[96].

Listing 4.3: Example sentence to illustrate the difference between NED and WSD

Thomas and Arjen are strikers playing in Munich.

The sentence in Listing 4.3 makes clear how interwined but also different the two approaches are. *Striker* and *play* are polysemous words which can be disambiguated by selecting the soccer game playing sense of the two words in a dictionary. But *Thomas* and *Arjen* are just partial mentions, which have to be linked to the appropriate entries in an external resource, that is *Thomas Müller* and *Arjen Robben*. Furthermore, both approaches differentiate in terms of the kind of inventory used for the linking process. WSD usually uses either dictionaries or thesauri, while NED utilizes knowledge bases such as encyclopedias or ontologies[96]. These differences may be the reason why the research community has so far approached the two tasks separately.

On the other side, current research in knowledge acquistion is trending towards the seamless integration of encyclopedic and lexical knowledge into structured language resources[73]. BabelNet[24][102] reinforces this finding. Hovy et al. [73] establish the hypothesis, that lexical knowledge used in WSD is also useful for tackling NED, and vice versa. Going further, even though NED and WSD differ from each other, the techniques used by them may be shared as well. Having the goal of this work in mind, obviously WSD approaches may be feasible as well. For that reason not just NED approaches are described in this chapter, but also WSD methodologies.

Depending on the specific task different knowledge based NED procedures exist. Due to the recent collaborative creation of large semi-structured resources, such as Wikipedia, and structured knowledge resources built from them[73], such as BabelNet[102], DBpedia[8] or YAGO[72], NED based on these resources has emerged[110]. Hereby the recognized NEs are linked to the most suitable entry in such a knowledge base. Moreover, such a knowledge base can be also an ontology or in particular for WSD dictionaries or thesauri. Concerning the disambiguation, different methodologies exist. As with almost

[24] `http://babelnet.org`

each NLP task, ML techniques exist. It can be distinghuised similar to NER between SML, SSML, and USML. However, Chapter 2 showed, that not much research has been done in terms of NED via SSML. In the following sections, different techniques to tackle NED (and WSD) are described. Furthermore, the templated NER approach from Section 4.1.4 can be enhanced to enable templated NED (Section 4.2.3).

4.2.1 SML-based NED

As with SML in NER, a corpus needs to be available for training. For NED it has to be a disambiguated corpus. The training data consists of a set of exemplars where each occurrence of the ambiguous word w is annotated with a semantic label, it's contextual sense s_k. Due to this, supervised disambiguation is an instance of statistical classification. Hence a classifier has to be build which correctly classifies new cases based on their context of use c_i[84].

Table 4.3: Notational conventions for SML-based NED

Symbol	**Meaning**
w	an ambiguous word
$s_1, ..., s_k, ..., s_K$	senses of the ambiguous word w
$c_1, ..., c_i, ..., c_I$	contexts of w in a corpus
$v_1, ..., c_j, ..., c_J$	words used as contextual features for disambiguation

An overview of this notation used for the remainder of this section is shown in Table 4.3. Researchers came up with quite a few approaches to address supervised NED. Two techniques being also highly relevant for statistical language processing in general[84], are introduced in the following. The *Bayesian classification* by Gale et al.[57] as well as the *Information Theory* proposed by Brown et al.[28]. Afterwards, a system developed by Cardellino et al.[31] is introduced.

4.2.1.1 Bayesian Classification

The Bayesian classification handles the the context of a word as a bag-of-words. The bag-of-words model is a simplified representation of text used in NLP and IR a lot. The structure of a text (such as a sentence or a document) is ignored in a bag of words. The multiplicity is kept, while disregarding any

grammar or word order[134]. Therefor it integrates information from many words in the context window[84]. Each word in the context of an ambiguous word w contributes potentially useful information for the disambiguation task. A Bayesian classifier performs no feature selection, but combines the evidence from all features[57].

$$P(s'|c) > P(s_k|c) for s_k \neq s' \tag{4.3}$$

The *Bayes decision rule* seen in Equation 4.3 is applied by a *Bayes classifier* when choosing a class. Bayes decision rule rule minimizes the probability of error[48, p. 10-43]. This is true because for each individual case it chooses the class with the highest conditional probability and hence the smallest error rate. The value of $P(s_k|c)$ is usually unknown, but can be computed using *Baye's rule* shown in Equation 4.4

$$P(s_k|c) = \frac{P(c|s_k)}{P(c)} * P(s_k) \tag{4.4}$$

Hereby $P(s_k)$ is the prior probability of sense s_k, the probability that an instance of s_k is present while knowing nothing about the context. The evidence about the context updates $P(s_k)$ in form of the factor $\frac{P(c|s_k)}{P(c)}$, which results in the posterior probability $P(s_k|c)$. Since only the selection of the correct class if of interest, the classification can be simplified by eliminating $P(c)$. Using logarithms of probabilities makes the computation simpler as well.

$$\begin{aligned} s' &= \arg\max_{s_k} P(s_k|c) \\ &= \arg\max_{s_k} \frac{P(c|s_k)}{P(c)} P(s_k) \\ &= \arg\max_{s_k} P(c|s_k) P(s_k) \\ &= \arg\max_{s_k} [log P(c|s_k) + log P(s_k)] \end{aligned} \tag{4.5}$$

Now the goal is to assign w to the sense s' where Equation 4.5 applies. The classifier described here based on Gale et al. is an instance of the *Naive Bayes* classifier. Naive Bayes is broadly used in ML because of its efficiency and its

ability to combine evidence from a large number of features[94]. The bag-of-words model is pretty important right here. Naive Bayes is only applicable if the state of the world that we base our classification on is described as a series of attributes[84]. Since the context of a word w is represented as a bag of words, where each word v_j that occurs in the context is in this bag, the classifier is applicable.

$$P(c|s_k) = P(v_j|v_j inc|s_k) = \prod_{v_j inc} P(v_j|s_k) \tag{4.6}$$

The Naive Bayes assumption (Equation 4.6) that attributes used for description are all conditionally independent has two consequences. First, all the structure and linear ordering of words within the context is ignored. Second, the presence of one word in the bag is independent of another[84]. However, this is clearly not the case. Obviously, the Naive Bayes assumption is not suitable if there are strong conditional dependencies between attributes. Nonetheless, there is a large number of cases in which it performs well, because the dependencies between attributes is not large enough[45].

$$s' = \arg\max_{s_k}[logP(s_k) + \sum_{v_j inc} logP(v_j|s_k)] \tag{4.7}$$

This leads to the modified decision rule in Equation 4.7. Hereby $P(v_j|s_k)$ and $P(s_k)$ are computed via Maximum-Likelihood estimation from the labeled training corpus.

$$\begin{aligned} P(v_j|s_k) &= \frac{C(v_j, s_k)}{C(s_k)} \\ P(s_k) &= \frac{C(s_k)}{C(w)} \end{aligned} \tag{4.8}$$

Equation 4.8 reveals this estimation. Hereby $C(v_j, s_k)$ is the number of occurrences of v_j in a context of sense s_k in the training corpus. $C(s_k)$ is the number of occurrences of s_k in the training corpus and $C(w)$ is the total number of occurrences of the ambiguous word w. Having this mathematical model in mind, an algorithm for a Naive Bayes classifier can be created[84].

Listing 4.4: Algorithm for disambiguation according to Naive Bayes

```
1  // Training
2  for all senses s_k of w do
3    for all words v_j in the vocabulary do
4      P(v_j|s_k) = C(v_j,s_k) / C(v_i)
5    end
6  end
7  for all senses s_k of w do
8    P(s_k) = C(s_k) / C(w)
9  end
10
11 // Disambiguation
12 for all senses s_k of w do
13   score(s_k) = logP(s_k)
14   for all words v_j in the context window c do
15     score(s_k) = score(s_k) + logP(v_j|s_k)
16   end
17 end
18 choose s' = arg max score(s_k)
```

Such an algorithm is revealed in Listing 4.4[84]. According to Gale and Church[58] as well as to Yarowsky[141], this algorithm reaches a precision of 90% for six ambiguous nouns.

4.2.1.2 Information-theoretic Approach

Compared to the Bayesian classification, an information-theoretic approach looks only at one informative feature in the context. However this feature is carefully selected from a large number of potential *informants*[84]. Obviously that is a huge advantage in comparison to Bayes classifier, which assumes the independence of all words in the context of word w. Brown et al.[28] reported indicators for French ambiguous words.

Table 4.4 shows three examples of their findings. Its object is a good indicator for the verb *prendre*. *Prendre une mesure* translates as *to take a measure*, while *Prendre une décision* means *to make a decision*. Likewise the word immediately left to *cent*, as well as the tense of the verb *vouloir* are proper indicators for these two words. In order to make a proper use of an informant, a categorization of the sense it indicates needs to be performed. Going back

Table 4.4: Highly informative indicators for three ambiguous French words

Ambiguous word	**Indicator**	**Examples: value → sense**
prendre	object	mesure → to take décision → to make
vouloir	tense	present → to want conditional → to like
cent	word to the left	per → % number → money value

to the previous example, such a categorization would be that *mesure* indicates *to take* and *décision* refers to *to make*. Figure 4.5 illustrates the Flip-Flop algorithm used by Brown et al. for that purpose[84].

Listing 4.5: Flip-Flop algorithm to find indicators for disambiguation

```
1 find random partition P = {P_1, P_2} of {t_1, ..., t_m}
2 while(improving) do
3   find partition Q = {Q_1, Q_2} of {x_1, ..., x_n}
4     that maximizes I(P;Q)
5   find partition P = {P_1, P_2} of {t_1, ..., t_m}
6     that maximizes I(P;Q)
7 end
```

Let $t_1, ..., t_m$ be the translations of the ambiguous word, and $x_i, ..., x_n$ the possible values of the indicator. $I(P;Q)$ is the mutual information.

$$I(P;Q) = \sum_{p \in P} \sum_{q \in Q} p(p,q) log \frac{p(p,q)}{p(p)p(q)} \tag{4.9}$$

The definition of the mutual information is shown in Equation 4.9(for a detailed explanation please refer to Cover and Thomas[37, p. 20]). Manning[84] reports that "it can be shown that each iteration of the Flip-Flop algorithm increases the mutual information $I(P;Q)$ monotonically. Hence a natural stopping criterion for that algorithm is that the mutual information does not increase anymore.

Going back to the example already discussed in this section, *prendre* shall be translated based on its object. Assuming $\{t_1, ..., t_m\} = \{take, make, rise, speak\}$ and $\{x_1, ..., x_n\} = \{mesure, note, exemple, dcision, parole\}$, the initial partition P of the senses might be $P_1 = \{take, rise\}$ and $P_2 = \{make, speak\}$[28,

p. 267]. Even though it depends on the the particular data used, which partition Q of the indicator values gives maximum mutual information ($I(P;Q)$), it is assumed that *prendre* translates by *take* when occurring with the objects *mesure, note, and exemple*, and translates by *make, speak, and rise* when occurring with *décision, and parole*. With that assumption, Q_1 will maximize the mutual information. An incorrect decision is only made when *prendre la parole* is translated as *rise to speak*, however this cannot be avoided since *rise* and *speak* are in two different partition groups. In the next two steps of the algorithm, P is as $P_1 = \{take\}$ repartitioned and P_2 as well as Q remain the same. The partition for *take* thus is always true. When a distinction between the other translations *make, rise, and speak* is wanted, more than two senses would have to be considered. This is not possible with the Flip-Flop algorithm presented in Listing 4.5[84]. The algorithm desinged by Brown et al.[27] is required for that purpose.

As soon as an indicator and a particular partition of its values has been determined by the Flip-Flop algorithm, disambiguation is straight forward. First, the value of x_i needs to be determined for the occurrence of the ambiguous word. Second, if x_i is in Q_1, the occurrence is assigned to sense 1, otherwise to sense 2[84]. Even though in this example, the information-theoretic approach is used for a translation problem, this can be adapted to typical NED problems as well.

4.2.1.3 Approach to NED of Cardellino et al.

Section 4.1.3.1.1 already introduced the NER classifier by Cardellino et al[31]. This section briefly describes their implementation of a NED system, which links NEs to YAGO URIs. The training corpus constituted 174.913 entities, which was too big to train a classifier directly[31]. Hence, they use a two-step classification pipeline. Using the NER classifier, they classify each mention as its most specific class in their own ontology. For each of these classes, they train a classifier to identify the correct YAGO URI, using only the URIs belonging to that specific class, detected in step one. For that reason, they build several classifiers, each of them trained with a reduced number of labels. The training of the two-step pipeline is defined as follows: (1) assign to each mention its ground truth ontology label, (2) split the dataset into train/test/validation, (3) for each assigned ontology class (3.1) build new train/test/validation datasets

by filtering out mentions not tagged with this class, and (3.2) train and evaluate a classifier with the new train/test/validation data sets. Afterwards, the actual linking is done by following these steps: (1) for each instance, assign a NE class to it using a previously trained NER classifier, and (2) select the classifier assigned to the class, and use it to obtain YAGO URI prediction of the instance. In terms of NED, Cardellino et al. only implemented the described algorithms as a neural network[31].

The NER of Cardellino et al. was evaluated by the test portion of their Wikipedia corpus as well as on a corpus of judgments of the ECHR. However, annotations on the level of entities has not been consolidated in the corpus of judgemtns and hence, NED was only evaluated on the Wikipedia corpus[31]. When measuring the performance of the whole pipeline, from NER via NED to the YAGO URIs, a F_1 of 0.16 is reached, while using ground truth for NER, a performance of F_1 0.45 is achieved. This supports the assumption, that NED for the legal domain still needs further research.

4.2.2 USML

The approaches introduced during the last section require for disambiguation basic lexical resources, a small training set or a few collocation seeds. It may seem little to ask for, but there are situations in which even such a small amount of data is not available. In particular when dealing with specialized domains, such as the legal domain, there may be no sufficient resources available[84]. This is because general dictionaries or knowledge bases are less useful for domain-specific tasks. Not just the domain knowledge needs to be specific for that domain, but also the algorithm may be adapted[84]. Having a legal ontology containing mostly legal terms, a generic ontology-based disambiguation algorithm would therefore be of little use. Such a knowledge base or training data can't be produced quickly on demand. Hence there is an increasing number of scenarios where outside sources of information are not available for disambiguation[84], this is the case in particular for the legal domain[12, 117]. That is the reason for USML-based NED approaches.

However, completely unsupervised disambiguation is not possible in terms of sense tagging. An algorithm labeling occurrences of a word to one sense or another, can't work without supervision or an external information source.

Differentiation of senses is possible though in a complete unsupervised fashion. Given a ambiguous word, an algorithm can collect all existing contexts, before it clusters them into different groups. As a result, it can be told whether two occurrences of a word mean the same or not, but it can't be told what the actual meaning is[84]. Several so called discrimination algorithms has been developed, like the one by Schütze[119]. For the goal of this work though, USML approaches in terms of NED are rather non beneficial.

4.2.3 Templated

Section 4.1.4 already briefly introduced the templated NER approach, which is developed within the scope of this study. Having a template of a contract, one could create a semantic model with regard to the template. To be more precise, a template consists of various placeholders, where the actual content is inserted during the contract creation process. A semantic model of such a contract, can be created while adding each placeholder to the model (as a type or an attribute). Going even further and regarding the placeholder names in the model, a linking is already created. Of course, the linking is established manually and this is basically just picking up the low hanging fruits, but this enables the straight process from NER, via NED towards a populated semantic model of a contract. This idea is implemented and also technically back-lit in Chapter 5.1.4.

4.2.4 Comparison

Various approaches to NED were discussed in the previous sections. Now, the different methodologies are compared, by using different criteria. Each criteria is assessed with either *low*, *medium*, or *high*, whereby the assertion depends on the actual criteria. Furthermore, *none* can be used if the criteria is not applicable. Table 4.5 summarizes this comparison.

USML-based NED suffers portability issues on both, language level and domain level, as well as SML approaches. Due to the language portability, which is achieved by translating the templates, the templated approach already seems to be a better fit. The latter method obviously does not need any training data in comparison to SML. USML does not need training data either, however it

Table 4.5: Comparison of NED approaches

Criteria	SML	USML	Templated
Language Portability	Low	Low	High
Domain Portability	Low	Low	Low
Required Training Data	High	None	None
Expert Knowledge	Medium	High	Low
Manual Work	Medium	Medium	Low
System Complexity	High	High	Low

is not possible yet, to utilize USML for NED purposes. Hence, USML is not interesting for an implementation in this work as of now. Of course, SML needs training data, which is one of the major problems with German legal data [31]. Hence, in the course of this work, only an approach to templated NED shall be implemented.

4.3 Involved Systems

The prototypical implementation to semantically analyze and structure legal contracts, which is implemented in the course of this thesis, is not a standalone application. Reusability is quite important in software engineering and even more important in research. Research is an ongoing process, involving various different researchers. Exchange of information but also of recent work thus is crucial [138]. Due to this, the prototypical implementation is designed as a software component. The chair of "Software Engineering for Business Information Systems" already has developed tools and software for legal informatics as well as for social software engineering. Two systems or frameworks developed by the chair, serve as a foundation for this work. The following two sections describe the two involved systems in closer detail.

4.3.1 Lexia Framework

Lexia[25] is a collaborative web based "data science environment for semantic analysis of German legal texts"[137] with the goal to analyze legal texts from

[25]Further information about Lexia as well as about the respective research project Lexalyze can be found at `https://wwwmatthes.in.tum.de/pages/1rvivk51a20k4/Lexalyze-Interdisciplinary-Research-Program`

different sources regarding their linguistic structure. The chair of "Software Engineering for Business Information Systems" at the TU München is developing Lexia as a research approach of tailoring generic NLP components to the domain of legal data science. "To achieve highest accuracy in terms of prediction and recall"[137], this tailoring process is necessary.

Inspired by the latest development in computer science and in particular in the field of *Artificial Intelligence* (AI), Lexia has been developed to support legal tasks. Those tasks are very knowledge-, data- and thus also time intensive[137].

Apache UIMA, developed by IBM and also used in IBM Watson, conduces as a reference architecture.

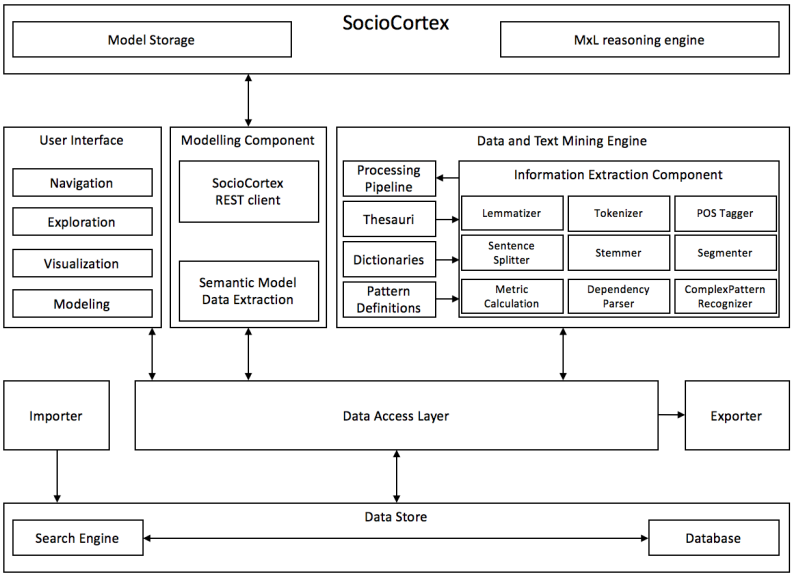


Figure 4.4: Architecture of Lexia

Source: own illustration based on [137]

Figure 4.4 shows the main components: an *exporter*, an *importer*, a *data storage* along with an *access layer*, a *text-mining engine*, and an *user interface*. The application is based on a Java back-end by utilizing the web application framework *Play*[26]. The data storage is handled by two parallel and independent platforms. Firstly an *Elasticsearch*[27] server is used to guarantee efficient

[26] http://www.playframework.com
[27] http://www.elastic.co/de

access to text data. Secondly, a *SocioCortex*[28] instance may be used for storage capabilities as well.

Legal texts in several formats such as PDF, HTML or XML can be imported into the Elasticsearch database of Lexia, using the *Importer*. The *Importer* supports a huge range of different sources for the import. The files can be stored locally, but also distributed in the internet on publishers such as *Beck-Online*[29] or *Rechtsprechung im Internet*[30]. During the import, the system detects the type of the legal document. Yet *laws*, *judgments*, *patents*, *contracts* and *miscellaneous* are supported, whereas the latter one acts as a backup document type.

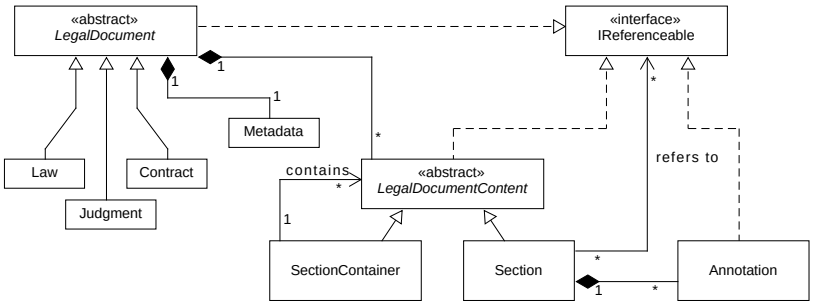


Figure 4.5: Data model of Lexia

Source: [137]

LegalDocument is the abstract base class of all document types, from which all specializations inherit. Figure 4.5 shows the data model of Lexia, which is tailored to the structure of legal documents. Each object contains *Metadata* with additional information about the object. The *LegalDocumentContent* is implementing a composite pattern[129]. In order to map the complex and nested structure of legal documents such as laws or contracts, the content is stored in *Sections*. For some document types, this structure may be already suitable. However, for arbitrary nested documents, *SectionContainers* exist. Such a container can hold one to many *Sections*. Furthermore a *SectionContainer* is able to contain zero to many *SectionContainers*. This perfectly suits the structure of most legal documents. The last object in the data model are *Annotations*. They can enrich *Sections* with further information. *Annotations*

[28] `http://www.sociocortex.com`
[29] `http://beck-online.beck.de/Home`
[30] `http://www.rechtsprechung-im-internet.de`

have a certain type and reflect the outcome of the *Data-* and *Text-Mining Engine*.

The heart of Lexia is the *Data-* and *Text-Mining Engine*. As previously mentioned, it is built on the Apache UIMA. The major component within that Engine is the *Information Extraction Component*. It has the ability to extract and annotate semantic information in legal texts. This is achieved by the support of dictionaries and pattern definitions such as regex and Apache UIMA Ruta scripts. By use of the rule language Ruta, patterns can be described in an easily maintainable and reusable way.

The aforementioned *Information Extraction Component* can be instanced several times during runtime. Hence the component can represent various tasks such as, sentence splitter, tokenizer, POS tagger or NE recognizer. The Apache UIMA utilizes the use of pipelines to process legal texts. The different instances of the *Information Extraction Component* can be concatenated and executed in one flow.

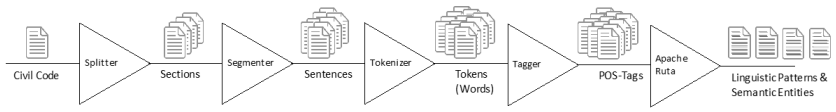


Figure 4.6: Processing pipeline for determining linguistic patterns with Apache UIMA and Ruta

Source: [136]

The concept behind these pipelines is shown in Figure 4.6. Due to that efficient processing of legal documents into *linguistic Annotations* as well as *semantic annotations* is enabled. *Linguistic annotations* are for instance the tagging of a sentence as a sentence or typical POS tagging. On the other side *semantic annotations* can be the recognition of a text passage as a legal definition. Figure 4.7 illustrates these different types of annotations.

The *User interface* (UI) of Lexia enables simple access to the documents for the end user. Tasks such as importing legal texts or examining the results of a processed pipeline is easy to accomplish as well. As long as the user has some legal background, a user can easily select a specific pipeline, set it up by means of different parameters, and eventually run it. Moreover the UI of Lexia offers various other functionalities that are not relevant for this thesis

Figure 4.7: Different annotation types shown in the Lexia user interface

Source: Own screenshot

and thus not described here. However, some important UI elements are shown and explained in Chapter 5.2.

In summary, Lexia is a powerful collaborative environment for analyzing legal, textual data linguistically and semantically. Nevertheless, in its current state, this knowledge is mostly applied on laws and judgments. The different analysis options describe here, can be applied on legal contracts as well. However, this is not that helpful in order to reveal the tailored and relevant information of a legal contract in a structured way.

To create this structured summary of a legal contract and to retrieve the relevant semantic information, a prototypical implementation in the form of a software component integrated into Lexia is implemented during that work. After the next section, Section 4.4 collects functional and non-functional requirements in order to develop such a component. Based on those requirements, Section 4.5 comes up with a suitable software architecture.

4.3.2 SocioCortex

SocioCortex is the second platform used as a data storage of Lexia. It is described in this section, due to the fact that it has been developed by the chair of "Software Engineering for Business Information Systems" of the TU München as well. SocioCortex is a hybrid wiki system. This kind of hybrid wiki system was first introduced my Matthes et al. [86] with the goal to create a lightweight and structured data management system on top of a unstructured

wiki system. When structuring wiki pages, usually users have to learn a new and complex semantic language. By its tailored design, this new approach has the goal to avoid such an issue. Hereby either the data import comes first (bottom up), or the creation of a data model (top down). Afterwards the user can be guided into a consistent and well defined data model through the addition of constraints[86].

Reschenhofer et al. [113] evaluated this new approach in different projects. During their evaluation, they adapted the approach which led to the meta model shown in Figure 4.8.

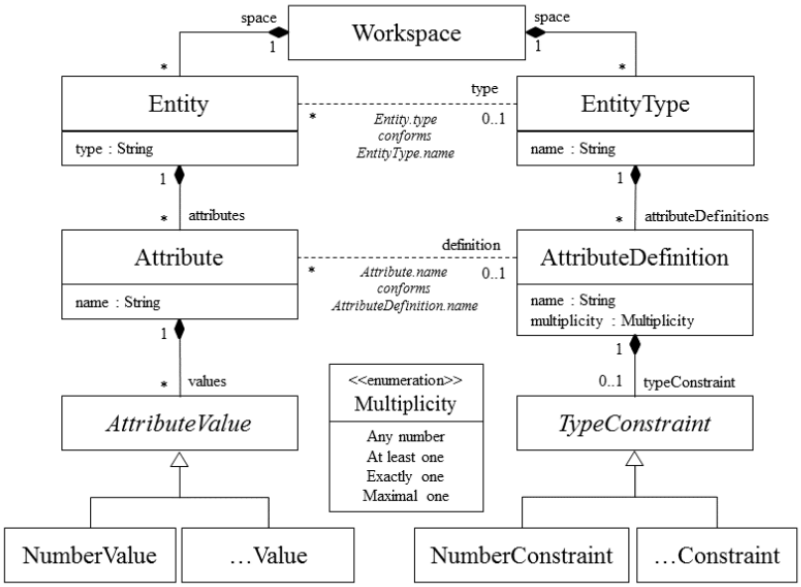


Figure 4.8: Meta model of the SocioCortex

Source: [106] based on [113]

Workspace is the root node and can contain several *Entities*. *Entities* are loosely coupled with an appropriate *Entity Type* by its type name. An *Entity* is used to represent an instance of a given *EntityType*. It can have many *Attributes* of a given *AttributeDefinition*. They are also loosely coupled by the *AttributeDefinition* name, such as *Entity* and *EntityType*. Each *Attribute* can have different *AttributeValues* that are of a specific value type (e.g. *NumberValue*, *TextValue*). *AttributeDefinitions* have a multiplicity which can either be any number, at least one, exactly one or maximal one.

In order to simplify the access of other software engineers, a comprehensive REST Application Programming Interface (API)[31] was added. By utilizing the REST API, all elements of the here described meta model can be created, modified, queried and deleted. For the data exchange, JSON is used.

Lexia initially used SocioCortex as a data storage, before Elasticsearch was introduced. Due to the smart indexing of Elasticsearch and thus performance improvements, Lexia uses Elasticsearch for the most part in terms of data storage. However, some parts of Lexia, such as the modeling environment, still use SocioCortex. Furthermore, the platform is able to switch seamless between Elasticsearch and SocioCortex.

When structuring legal contracts, a model of a specific type of contract must be defined. In the course of this thesis, the models are created in the SocioCortex. Furthermore, the instances of a certain model are handled by the SocioCortex as well. Chapter 5 deals with the details about that.

4.4 Requirements Analysis

After familiarizing with state of the art technologies, the first step of developing a software component to semantically analyze and structure legal contracts is to conduct a requirements analysis. As described in the previous chapter, the requirements have been elicited by a literature review. Moreover, the current state of Lexia was minded as well for the elicitation of the requirements.

This section presents the essential requirements of the prototypical implementation and it is explained why the software component should fulfill these requirements and what consequences they implicate.

In order to illustrate the relevance, the keywords *shall*, *should*, and *may* specified by the ISO[32] are used: *shall* indicates a requirement, *should* a recommendation, and *may* is not mandatory, but useful to have.

[31] The documentation is available at `http://www.sociocortex.com/documentation/`

[32] https://www.iso.org/foreword-supplementary-information.html (last accessed: August, 31th 2017)

The following requirements are divided into *Functional Requirements* (FR) and *Non-functional Requirements* (NFR). Section 4.4.1 discusses the FRs and Section 4.4.2 deals with the NFRs. At the end, Section 4.4.3 provides an overview of the requirements in a tabular form. In this section, the prioritization takes places as well, even though the compliance to the ISO standard already gives a good indicator about the relevance of each requirement.

4.4.1 Functional Requirements

The FRs affecting the behavior of the prototypical implementation, and hence the design, are described in the following.

FR01: Statistical ML based NER

As seen in Section 4.1, there are two types of NER approaches (Knowledge-based and Statistical ML), enhanced by an approach, which is developed in the course of this work (Templated NER). The software component *shall* be designed to perform and evaluate statistical ML based NER.

FR02: Knowledge-based NER

The prototypical implementation *shall* be designed to perform and evaluate knowledge-based NER.

FR03: Templated NER

An approach to NER based on templates is implemented during the course of this work. The software component *shall* be designed to perform and evaluate templated NER.

FR04: Supervised NED

In order to link NEs to types of the semantic model, defined by the user, a supervised approach *may* be implemented.

FR05: Unsupervised NED

The prototypical implementation *may* include an unsupervised technique to link NEs to semantic roles within a model.

FR06: Dictionary-based NED

An approach based on a dictionary-base *may* be implemented within the software component.

FR07: Templated NED

FR03 describes a requirement to include the templated NER approach, developed during this work. The prototypical implementation *should* implement a functionality to link NEs, identified by the templated NER methodology, to semantic functions.

FR08: Use of Apache UIMA Pipelines for NER

It *shall* be possible to execute the different approaches to NER (see FR01, FR02 and FR03) isolated from each other. For that reason, each NER approach *shall* be integrated into an Apache UIMA pipeline. It also *shall* be possible to execute the NED approaches seamlessly after a certain NER approach. Hence the approaches to NED (see FR04, FR05, FR06 and FR07) *shall* be implemented as Apache UIMA pipelines as well.

FR09: Combine NER and NED Pipelines

FR08 already requires the use of Apache UIMA pipelines. The software architecture *should* allow the combination of different NER and NED pipelines, to create NERD pipelines.

FR10: Create Contract Models

The results of a NED pipeline must be stored in the back-end. For that reason, contract models are used (see NFR07). The user *shall* be able to create such a contract model in the UI.

FR11: Update and Change Contract Models

In the lifecycle of a legal contract, it may be changed from time to time. Hence, it *should* be possible to update an existing contract model.

FR12: Delete Contract Models

Once a contract model is not needed anymore, the protoypical implementation *may* support the deletion of such a contract model.

FR13: Use of Lexia's Drafted Legal Documents

As describe in Section 4.3.1, Lexia comes with different document types. One of the types is a *DraftedDocument*. The software component *shall* use this type for the semantic analysis and restructuring.

4.4.2 Non-functional Requirements

After defining the FRs, NFRs must be elicited as well. The NFRs are defined and discussed in this section. NFRs influence the system's architecture heavily.

NFR01: Simple UI

It *shall* be easy for the user to choose and run a NER or NERD pipeline. The user *shall* easily see the results of a pipeline and thus face the structured contract information. There *should* be a clear differentiation between the different steps in the process of structuring an unstructured legal contract.

NFR02: Maintainability of Software Architecture

It *should* be easy for other developers to familiarize with the code in order to maintain the software component. Hence, the software architecture *should* be well defined and structured in a way supporting maintainability.

NFR03: Extensibility of Software Architecture

The adding of new functionality like additional or different NER approaches, as well as NED approaches *shall* be possible without considerable refactoring. In particular creating new pipelines *shall* be easy for other developers.

NFR04: Reusability of Software Architecture

The software architecture *should* support the reuse of the components.

NFR05: Use GermaNER for Statistical ML NER

As described in Section 4.1.3.1.2, GermaNER is an open source and thus freely available statistical ML-based NER implementation. GermaNER *shall* be used as one NER approach.

NFR06: Use DBPedia for Knowledge-based NER

Section 4.1.2 discussed DB[]edia as a proper source for knowledge-based NER. DBPedia is freely available and hence *shall* be used as one NER approach.

NFR07: Reuse the Models and Structure from the Semantic Model Component

Oppmann[106] created a semantic model component for Lexia during his master's thesis. The model class and as much as possible other parts and pieces *may* be reused.

NFR08: Incorporate Elasticsearch for the Storage

Lexia is using an Elasticsearch server to index the documents and annotations, as described in 4.3.1. The prototypical implementation *shall* use Elasticsearch for storing the contract models. This is also important to keep the data handling within Lexia consistent.

NFR09: Incorporate SocioCortex for the Storage

Section 4.3.2 described the SocioCortex platform in detail. The actual instances of the contract models *should* be handled by the SocioCortex. Due to this, NFR07 is supported as well.

NFR10: Utilize Lexia's existing Pipeline Architecture

The data- and text-mining engine of Lexia already implements various Apache UIMA pipelines for text processing and semantic analysis. The software architecture *shall* reuse this architecture.

4.4.3 Summary and Prioritization

A student's work like this one is limited in time and resources. Thus, after all FR and NFR are elicited, they must be prioritized. When designing the software architecture, as many as possible requirements are regarded. However, there may be some requirements with a lower priority, being disregarded partially.

Table 4.6 provides an overview of all FRs and NFRs, elicited during the previous two sections. The requirements are already prioritized in Table 4.6.

The priority column reflects the ISO-keywords as a number, scaled from 1 to 5, where 5 indicates highest priority and 1 lowest priority:

- may $\hat{\approx}$ priority 1 - 2
- should $\hat{\approx}$ priority 3
- shall $\hat{\approx}$ priority 4 - 5

Table 4.6: Summary of all requirements

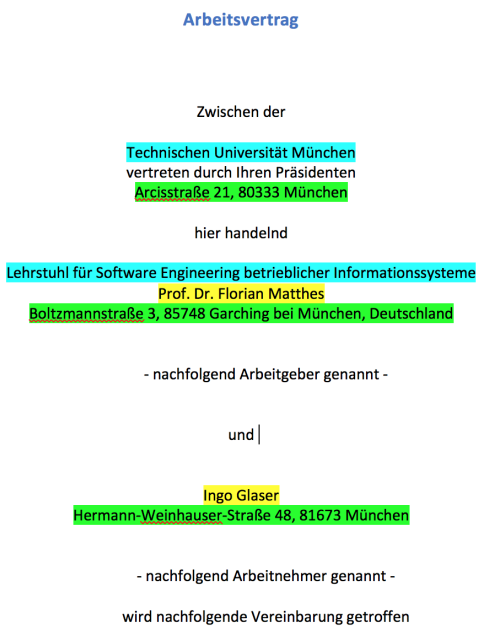
ID	Requirement	Priority
FR01	Statistical ML based NER	4
FR02	Knowledge-based NER	4
FR03	Templated NER	5
FR04	Supervised NED	2
FR05	Unsupervised NED	1
FR06	Dictionary-based NED	2
FR07	Templated NED	3
FR08	Use of Apache UIMA Pipelines for NER	5
FR09	Combine NER and NED Pipelines	3
FR10	Create Contract Models	5
FR11	Update and Change Contract Models	3
FR12	Delete Contract Models	2
FR13	Use of Lexia's Drafted Legal Documents	5
NFR01	Simple UI	5
NFR02	Maintainability of Software Architecture	5
NFR03	Extensibility of Software Architecture	5
NFR04	Reusability of Software Architecture	3
NFR05	Use GermaNER for Statistical ML NER	4
NFR06	Use DBPedia for Knowledge-based NER	4
NFR07	Reuse the Models and Structure from the Semantic Model Compoenent	2
NFR08	Incorporate Elasticsearch for the Storage	5
NFR09	Incorporate SocioCortex for the Storage	3
NFR10	Utilize Lexia's existing Pipeline Architecture	5

4.5 Architecture

After introducing related concepts and eliciting requirements, in this section the architecture of the prototypical implementation is designed. First of all, a conceptual overview revealing the process of semantically analyzing and structuring legal documents is presented. Afterwards the software components and data model of the implementation are described in greater detail. Eventually, the workflow of such a process is revealed, followed by an overview of the REST API.

4.5.1 Conceptual Overview

In order to achieve the goal of semantically analyzing and structuring legal contracts, a process consisting of NER and NED is defined. This concept serves as a reference for the actual implementation. For the explanation of this concept, the example of an employment agreement is taken.


Arbeitsvertrag

Zwischen der

Technischen Universität München
vertreten durch Ihren Präsidenten
Arcisstraße 21, 80333 München

hier handelnd

Lehrstuhl für Software Engineering betrieblicher Informationssysteme
Prof. Dr. Florian Matthes
Boltzmannstraße 3, 85748 Garching bei München, Deutschland

- nachfolgend Arbeitgeber genannt -

und

Ingo Glaser
Hermann-Weinhauser-Straße 48, 81673 München

- nachfolgend Arbeitnehmer genannt -

wird nachfolgende Vereinbarung getroffen

Figure 4.9: Example of an employment agreement

Source: Own illustration

The first step towards the extraction of semantic knowledge is the application of NER. The goal of this step is to extract all NEs in the agreement. The result of this task is illustrated in Figure 4.9. Blue highlighted phrases reflect organisations, green phrases locations, and yellow phrases persons.

Text	Named Entity	Role
Technische Universität München	Institution	Employer
Bernhard Waltl	Person	Employee
3 Monate	Time duration	Cancelation period
30. April 2018	Date	End of contract
...	...	...

Figure 4.10: Conceptual overview of the recognition and disambiguation process

Source: Own illustration

Once the NEs are recognized, the actual disambiguation can take place. Each contract needs to be modeled. For that reason, the prototypical implementation allows the definition of such a semantic model. For the sake of this example, assuming a model with roles like employer, employee, cancellation period and end of contract, Figure 4.10 includes both steps of the NER and NED process. The phrase *"Technische Universität München"* is recognized and classified as a *organization* in the first step. During the second step, the NE is linked to the respective role *Employer*.

This is the basic concept behind the software component, being implemented in the course of this thesis. Figure 4.11 depicts the described concept within a conceptual software architecture. It reflects the basic architecture of the semantic analysis component. The *Semantic Analysis Component* consists of two sub components, that is: (1) the *Named Entity Recognition* component, and (2) the *Named Entity Disambiguation* component. The former gets a *Contract* as input and performs NER on it. Optionally for the templated NER, it consumes a *Template*, too. This results in an *Annotated Contract*. This *Annotated Contract* is forwarded to the disambiguation component. Depending on the approach, a *Template*, external resources such as knowledge bases as well as AI is used to create the *Structured Contract*. The integration of this *Seman-*

tic Analysis Component into Lexia's architecture is described in the following section (4.5.2).

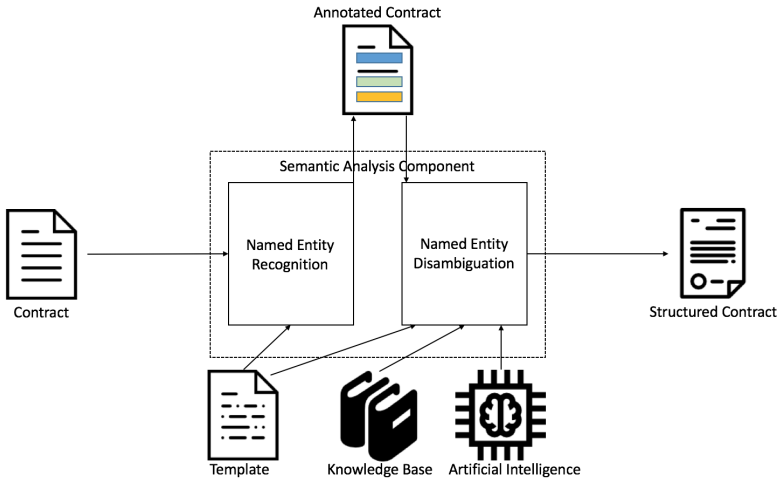


Figure 4.11: Conceptual architecture of the semantic analysis component

Source: Own illustration

4.5.2 Software Architecture

Figure 4.12 depicts the target architecture of the system. The new components which were implemented are highlighted in blue (cf. with Figure 4.4 in Section 4.3). Components by Oppmann[106] being reused and or adapted are shown in grey.

The main contribution of this work to Lexia is the *Semantic Analysis Component*, which consists of two sub components, namely the component for *NER* and for *NED*. Furthermore the UI was adjusted accordingly. The *NER Component* is responsible for performing the NER task, while the *NED Component* does so for NED.

The *Semantic Analysis Component* can connect to external resources such as DBpedia in order to perform its tasks. As already described in Section 4.3, Lexia's *Data and Text Mining Engine* is based on Apache UIMA. The components of this work shall be executed within an Apache UIMA pipeline as well. For that reason, the *Semantic Analysis Component* is intertwined with the *Data and Text Mining Engine*. Due to this, it can obtain knowledge

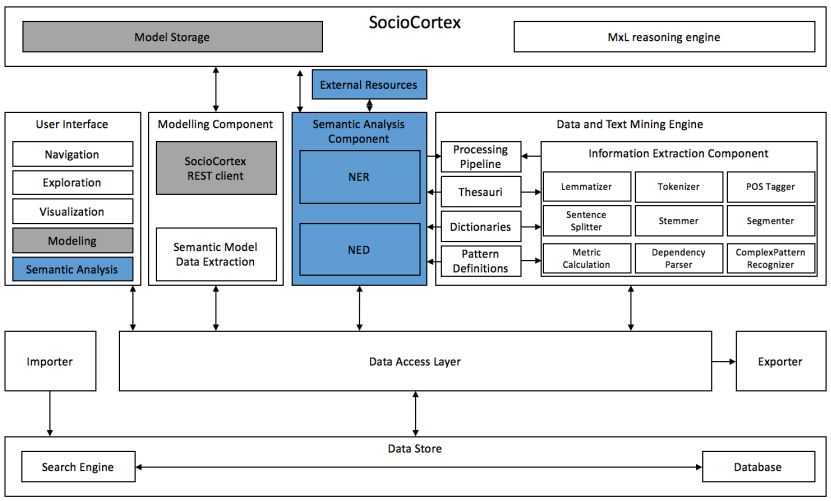


Figure 4.12: Target architecture of Lexia

Source: Own illustration

from the resources within the *Data and Text Mining Engine*, such as *thesauri*, *dictionaries* or *pattern definitions*.

NFR07 already demands the reuse of parts of the modeling components developed by Oppmann. In fact, this prototypical implementation uses the same concept for the storage of semantic models. First, models are stored in the Elasticsearch server, to be able to link them to the other domains such as *DraftedDocuments*. Second, the semantic information of a model is stored in the SocioCortex. In order to access SocioCortex for persisting the gained semantic information, the *SocioCortex REST client* is used. In terms of UI, parts from the *Modeling Component* are adapted. The UI components of this system are describe in Chapter 5.2.

Speaking about UI, the *Semantic Analysis UI* is purely implemented in HTML5, CSS3, and JavaScript, utilizing the AngularJS[33] framework. The parts being adapted from Oppmann, use JointJS[34], a diagramming library, to draw visual representations of the semantic models. Moreover, vis.js[35] is used to render the object diagrams.

[33] `http://angularjs.org`
[34] `http://www.jointjs.com`
[35] `http://visjs.org`

4.5.3 Mapping between Semantic Model Elements and SocioCortex Entities

The semantic model, which is created in order to structure a legal contract based on the extracted semantic information, shall be stored in the SocioCortex. For that reason, a mapping must be created which maps the components of the graphical model representation to entities of SocioCortex. Table 4.7 shows the approach of how this mapping was implemented already by Oppmann[106] and hence borrowed for this work. Nonetheless some mappings are superfluous for this work and thus are not discussed here.

Table 4.7: Mapping between semantic model elements and SocioCortex entities

Semantic model element	**SocioCortex entity**
Model	Workspace
Type	EntityType
Attribute	AttributeDefinition

This mapping naturally mirrors the structure of the semantic model onto the tree structure of SocioCortex entities. Figure 4.13 opposes the hierarchy of the semantic model elements in Lexia with their mapped entities from SocioCortex. *Attributes* are mapped to *AttributeDefinitions*, *Types* are interpreted as *EntityTypes* and the *Model* itself is mapped to the *Workspace*.

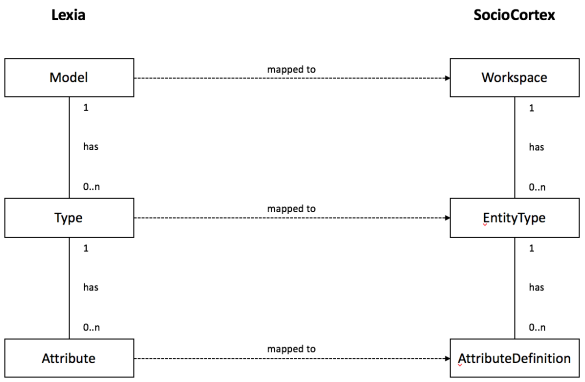


Figure 4.13: Tree hierarchy of semantic model elements and SocioCortex entities

Source: Own illustration based on[106]

4.5.4 Workflow Overview

The previous section gave an overview of the architecture of Lexia along with the component for the semantic analysis. This section provides a more profound insight into the interaction between the user and the system in order to perform a task for semantically analyzing and structuring legal contracts. The results of such a workflow are additional valuable information for the design of the required methods of the REST API presented in Section 4.5.5.

To conduct semantic analysis and structuring of legal contracts, two actors are required: (1) the user utilizing the UI and creating legal contracts for the (2) legal data science environment Lexia. In Figure 4.14 the interaction between the user and the system is illustrated. The arrows indicate the message flow along with their direction. The workflow consists of three main stages: (1) the preparation of the (bordered with blue) (2) NER (framed with orange) and the (3) NED (colored in yellow). Hereby the preparation stage is utilized twice, once for NER and another time for NED.

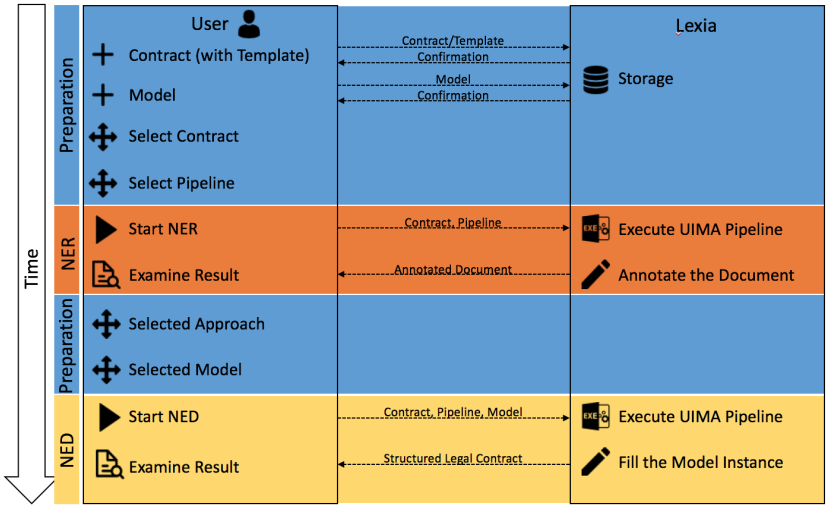


Figure 4.14: Conceptual workflow model of the semantic analysis

Source: Own illustration

The following paragraphs describe the three workflow stages in detail, starting from the top of Figure 4.14.

Preparation (1)

The preparation consists of six steps, whereas four steps are used to setup NER, and two steps are used to configure NED.

1. In order to perform NER as well as NED, a legal contract needs to be available in the system. The user creates such a contract, which is optionally based on a template. The system stores the contract in the Elasticsearch data storage.
2. For the disambiguation process, a semantic model is required. The user designs a model based on an existing contract by utilizing the UI. The model is saved in the Elasticsearch server, while the types and attributes are stored in the SocioCortex.
3. Before the NER task can be triggered, a contract needs to be selected. The NER performs its task on the selected document.
4. Due to the fact different NER approaches are integrated into this prototypical implementation, the user selects the desired technique.

NER

1. After everything has been prepared, the user initiates the NER task in the UI. On behalf of the user, the system executes the respective Apache UIMA pipeline, according to the setup. After the execution, the resulting annotated document is returned.
2. The user examines the annotated document and is able to see detailed information about the annotations.

Preparation (2)

1. In this step, the user selects an appropriate approach for the linking process, based on the selected NER pipeline. Hereby, the UI supports the user by pre-filtering the options. For instance, if the user has run a templated NER pipeline, only templated NED is available.

2. The recognized NEs are subject to be linked to a model, representing the processed contract. For that reason, the user selects an appropriate model.

NED

1. Once again, after the configuration is done, the user triggers the NED task. As a consequence, the system executes the related Apache UIMA pipeline.

2. After successfully running the pipeline, the model is populated with the semantic information gained during this step. The newly structured contract is returned to the UI, which allows the user to review the structured depiction of the contract.

4.5.5 REST API

For the communication between the frontend and backend of Lexia, a REST API is already in use. The REST API is a software architectural style that allows two systems or components to communicate with each other in a stateless fashion. It is the state-of-the-art technology for web applications[85]. A transferred message mainly consists of three components: a *Hypertext Transfer Protocol* (HTTP) method such as GET or POST, an URI, and some content type such as JSON or XML.

In what follows, the enhancement of Lexia's REST API, necessary to build the semantic analysis component developed in this work, is described in Table 4.8. This API enables the communication between the front-end and back-end. The structure and purpose of each request are described as well as the JSON nodes that must be part of the request body. The description of the body does not reflect the actual correct JSON architecture of the request, but provides a suitable indication what the actual request needs. This REST API also reflects the requirements defined in Section 4.4, and the workflow defined in Section 4.5.4.

In general, there are routes to trigger the execution of pipelines such as a NER pipeline, a route to retrieve model instances corresponding to a specific

contract, and a request in order to retrieve all models which are based on contracts. Furthermore, other routes of the existing REST API has been used in this work. Table 4.9 reveals these routes. These routes are mainly for the creation, update, and deletion of models.

Table 4.8: REST API for the semantic analysis component

Method	**URI**	**Purpose**	**Body**
GET	/api/legalDocument/drafteddocument/:did/modelInstances	Return all model instances corresponding to a specific contract	
GET	/api/semanticanalysis/pipeline/tempoary/germaner/*articleId	Run temporary GermaNER pipeline on an article	
GET	/api/semanticanalysis/pipeline/tempoary/dbpedianer/*articleId	Run temporary DBpedia pipeline on an article	
GET	/api/semanticanalysis/compare/diff/*articleId	Run temporary templated NER on an article	
POST	/api/model/contract	Return all models related to contracts	draw, start, length searchValue, order
POST	/api/semanticanalysis/pipeline	Execute a NER pipeline	documents, pipeline
POST	/api/semanticanalysis/entitylinking/templatedner	Execute templated NED	approach, modelId, document, entities

Table 4.9: Existing REST API routes used for the semantic analysis component

Method	**URI**	**Purpose**	**Body**
GET	/api/annotation/*aid	Retrieve a specific annotation	
GET	/api/model/:id	Retrieve a specific model	
POST	/api/model	Create a model	model
POST	/api/model/import	Import an external model	file
PUT	/api/model/:id	Update a specific model	model
DELETE	/api/model/:id/data	Delete all instances of a model	
DELETE	/api/model/:id	Delete a specific model	

5 Implementation Phase

In the previous chapter (4), the objectives and requirements as well as a suitable strategy in order to implement the prototypical implementation from this work were discussed. The next step is to actually implement it. The realization of this implementation is described in this chapter. The first section 5.1 deals with the back-end implementation before Section 5.2 explains the development of the UI.

5.1 Back-end

The enhancement of the Lexia back-end mainly accomplishes two tasks: (1) to perform NER, and (2) to perform NED. As already described in Chapter 4.3.1, Lexia is developed with the web application framework Play. Hence, a Java back-end takes over the main text mining functionalities. Furthermore the *Text and Data Mining Engine* of Lexia, utilizes the Apache UIMA reference architecture. NFR10 asks for the incorporation of Lexia's existing pipeline architecture. Consequently the new implementations are integrated into that existing architecture.

Section 5.1.3 deals with the implementation of the NER task, while Section 5.1.4 describes the NED development. Before that, Section 5.1.1 and Section 5.1.2 elaborate on the software pieces being necessary for this implementation.

5.1.1 Lexia's existing Pipeline Architecture

First of all, Lexia composes a variety of different pipelines for specific goals. For the purpose of managing all these pipelines, a *PipelineRepository* exists.

Listing 5.1: Excerpt of PipelineRepository to manage pipelines

```
1  public class PipelineRepository {
2      private static PipelineRepository pipelineRepository = new PipelineRepository();
3
4      private List<PipelineDescription> pipelines = Arrays.asList(
5          pipe("Advanced Pipeline", AdvancedPipeline::new),
6          pipe("Subject Pipeline", SubjectPipeline::new, false)
7          // More pipelines
8      );
9
10     // Constructor
11
12     public static PipelineRepository getInstance() {
13         return pipelineRepository;
14     }
15
16     public Pipeline getPipeline(int id) {
17         return pipelines.get(id).supplier.get();
18     }
19
20     // Getter and Setter
21     // Convenient methods for adding new pipelines
22 }
```

The *PipelineRepository* is shown in Listing 5.1. The class uses the singleton pattern[132] and holds all pipelines in a list. It offers convenient methods for adding new pipelines (not depicted in the listing above), various getter and setter methods (also not shown in the listing above) as well as different functions to retrieve a specific pipeline, such as the *getPipeline* method which return a pipeline based on its id. For the full source code of the *PipelineRepository*, please refer to Appendix A.1

All pipelines hold by that repository inherit from the abstract base class *Pipeline*. The source code of this class is partially shown in Listing 5.2.

Listing 5.2: Excerpt of the abstract base class for all pipelines

```
1  public abstract class Pipeline {
2      protected AnalysisEngine pipe;
3      protected JCas jCas;
4
5      public void setup(LegalDocument legalDocument, String[] rutaScripts) throws
           ResourceInitializationException, IOException, InvalidXMLException,
           CASException {
6          // Implementation
7      }
8
9      protected void initCas(LegalDocument legalDocument, String[] rutaScripts) throws
           ResourceInitializationException, IOException, InvalidXMLException,
           CASException {
10         // Implementation
11     }
12
13     // Convenient Methods
14
15     public PipelineResult process(Article article, String text) throws
           AnalysisEngineProcessException {
16         // Implementation
17     }
18
19     public PipelineResult processStandalone(Article article, String text, String[]
           rutaScripts) throws AnalysisEngineProcessException, CASException,
           ResourceInitializationException, InvalidXMLException, IOException {
20         // Implementation
21     }
22
23     public void preDocument(LegalDocument d) {
24     }
25
26     public void postDocument(LegalDocument d) {
27         // Implementation
28     }
29
30     public void preArticle(Article article, String text) {
31         // Implementation
32     }
33
34     public void postArticle(Article article) {
35         // Implementation
36     }
37 }
```

Again, for the full reference please refer to the Appendix A.2. The code shown in Listing 5.2 is the crucial one in order to understand the sense behind this class. After a pipeline has been retrieved from the *PipelineRepository*, its *setup* function is invoked. First, *setup* assembles the pipeline and assigns it to the *pipe* attribute. Second, the *jCas* object is initialized. After these steps a pipeline is basically already prepared to be executed. Nonetheless, the *Pipeline* class suggests different hooks to be completed on different stages within the pipeline execution. Preparation steps can be undertaken in *preDocument*, and *preArticle*, whereas the function is called before the processing of a document, respectively an article, starts. The same intention is behind the methods *postDocument*, and *postArticle*, but the action is performed after the corresponding stage. The *Pipeline* class furthermore offers different convenient methods (please also refer to the Appendix A.2). The pipelines developed in the course of this thesis shall comply to this structure and thus also inherit *Pipeline*.

In order to execute pipelines, a designated class *PipelineExecutor* exists. A nice feature of the class is its ability to handle various threads while being thread-safe. When executing a *Pipeline*, *PipelineExecutor* invoked initially the pipeline's *setup* function. Next it calls *preDocument* before it actually loops through all the articles. Hereby, before each article is processed, the *preArticle* function is called, while *postArticle* is called after processing the article. Once all articles are processed, eventually *postDocument* is called. The actual mechanics as well as the implementation is not relevant for this work. Due to this, the source code is just attached to the appendix of this thesis (Appendix A.3)

5.1.2 Lexia's Data Model for Legal Contracts

Figure 4.5 in Section 4.3.1 already introduced briefly Lexia's data model. For the purpose of this work, only the document type *DraftedDocument* is used. As the other legal document classes it inherits from the abstract base class *LegalDocument* which in turn inherits from the abstract superclass *Entity*. This class defines the basic interface of every entity within Lexia, that is an id and type attribute as well as methods for storage and deletion. Please refer to Appendix A.4.

Listing 5.3: Excerpt of abstract base class LegalDocument

```
public abstract class LegalDocument extends Entity {
    public final TreeMap<String, Function<LegalDocument, Double>> metricList =
        new TreeMap<>();
    public String annotationDataForDownload;

    private void initMetricList() {
        metricList.put("Word Count", MetricCalculation::countWords);
        metricList.put("Sentence Count", MetricCalculation::countSentences);
        // Other metrics
    }

    public TreeMap<String, Integer> totalAmountOfAnnotations = new TreeMap<>();
    public TreeMap<String, TreeMap<String, Integer>>
        totalAmountOfDistinctAnnotations = new TreeMap<>();

    protected String language = "de";
    public String title;
    private String ID;
    public String abbreviation;
    public Date creationDate;
    public Date promulgationDate;

    private String explicitNetwork;
    private int[][] implicitNetwork;

    public String annotationStructures;

    public List<ArticleContainer> articleContainers = new ArrayList<>();
    private List<Article> articles;

    public TreeMap<String, Double> legalDocument_metrics;

    public LegalDocument() {
        articleContainers = new ArrayList<>();
        legalDocument_metrics = new TreeMap<>();
        articles = new ArrayList<>();
        abbreviation = "";
        this.creationDate = new Date();
        this.promulgationDate = new Date();
        initMetricList();
    }
    // Convenient Methods
}
```

An excerpt of the abstract base class *LegalDocument* is depicted in Listing 5.3. The main purpose of this class is the definition of attributes required by all legal documents as well as convenient methods (such as getter and setter, determination of metrics, or to retrieve other meta information). Of course a legal document has an *ID*, being represented by the unique identifier provided from Elasticsearch. The attributes *title*, *language* and *abbreviation* are self-explanatory. The date by which a document has been created (imported or drafted) is stored in *creationDate*, while *promulgationDate* indicates the due date. Legal documents, in particular laws, often reference each other. When extracting these references, networks can be built. The attributes *explicitNetwork* and *implicitNetwork* capture such networks. A main feature of Lexia is the IE. Hereby the obtained information is depicted in form of annotations. In fact, an annotation is not directly linked to a document, but to an article. All articles of an document are stored in the list *articles*, while the article containers remain in the respective list *articleContainers*. Each annotation has a distinct type. All annotation types of the annotations belonging to a document (via the articles) are consolidated and kept in the *annotationStructures*. This attribute is heavily utilized by the UI in order to allow the user to select the annotations which should be displayed. For the analysis of legal documents metrics are very useful. These metrics are stored in the *metricList*, after they are determined.

Since this work semantically analysis and structures legal contracts, the respective class is intensely used. The class *DraftedDocument* does not introduce any new attributes, but includes plenty convenient methods. Furthermore, the functionality for storing a *DraftedDocument* (Listing 5.4), as well as retrieving it is implemented. Hereby various methods exist, supporting different retrieval criteria. The Elasticsearch server always return a map of key-value pairs. The function *documentFromMap* depicted in Listing 5.5 performs the mapping from such a map to the actual Java class.

Listing 5.4: Method to store contracts

```
public class DraftedDocument extends LegalDocument {
    // . . .
    @Override
    protected boolean saveEntityElasticsearch() {
        try {
            Map<String, Object> attributes = new HashMap<>();
            if (this.title != null && !this.title.isEmpty()) {
                attributes.put("Title", this.title);
            }
            if (this.creationDate != null) {
                attributes.put("CreationDate", this.creationDate);
            } else {
                attributes.put("CreationDate", new Date());
            }
            // More Attributes being stored
            if (this.isNewEntity()) {
                String uid = ElasticsearchServer.insert(this.SC_TYPE(), attributes);
                this.setID(uid);
            } else {
                ElasticsearchServer.update(this.SC_TYPE(), this.getID(), attributes);
            }
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
    // . . .
}
```

A map with strings as key and objects as value is created at the beginning of *saveEntityElasticsearch*. Then it takes each attribute of a *DraftedDocument* and inserts it into the map. This function is used for initially storing a contract as well as for updating it. Hence before the actual request to the Elasticsearch server can be sent, it is checked whether the entity is a new one or not. In either case the *attributes* map is forwarded to the Elasticsearch server to perform the storage function.

Listing 5.5: Method to populate a *DraftedDocument*

```
1  public class DraftedDocument extends LegalDocument {
2      // . . .
3      public static DraftedDocument documentFromMap(Map<String, Object> esMap) {
4          DraftedDocument document = new DraftedDocument();
5          try {
6              SimpleDateFormat dateFormat = new
                   SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
7              document.setID((String) esMap.get("id"));
8              if (esMap.containsKey("Title")) {
9                  document.title = (String) esMap.get("Title");
10             }
11             if (esMap.containsKey("CreationDate")) {
12                 document.creationDate = dateFormat.parse((String)
                       esMap.get("CreationDate"));
13             }
14             // Other Attributes being populated
15         } catch (Exception e) {
16             e.printStackTrace();
17         }
18         return document;
19     }
20     // . . .
21 }
```

As already mentioned, *DraftedDocument* offers a few methods to retrieve a contract from the database. Once the document is retrieved in form of a map with strings as keys and objects as value, the function *documentFromMap* is called within the respective retrieval method. First, an empty *DraftedDocument* is created, before it is populated with the corresponding attributes.

The information being provided in this section is sufficient in order to understand the parts of Lexia's data model, which are used in the course of this thesis. The following subsections deal with the actual implementation.

5.1.3 NER

This section describes the implementation of the NER task of this prototypical implementation. The requirements FR01, FR02, and FR03 ask for three different approaches to NER. During the conclusion of the different techniques

to NER in Chapter 4.1.5, three specific technologies have been identified. Section 5.1.3.1 deals with the implementation of a NER approach, incorporating GermaNER. Afterwards in Section 5.1.3.2 the development of a NER task, utilizing DBpedia Spotlight is described. Eventually templated NER is implemented during Section 5.1.3.3.

5.1.3.1 GermaNER

In Chapter 4.4.2, the NFR05 demands the incorporation of GermaNER as a statistical NER approach. After discussing GermaNER in Chapter 4.1.3.1.2, this section describes the prototypical implementation of it.

Lifecycle of *GermaNERPipeline*

As in Section 5.1.1 already mentioned, each pipeline needs to implement the abstract pipeline class. That section also described the typical processing life-cycle of such a pipeline. This life-cycle is again depicted in Figure 5.1.

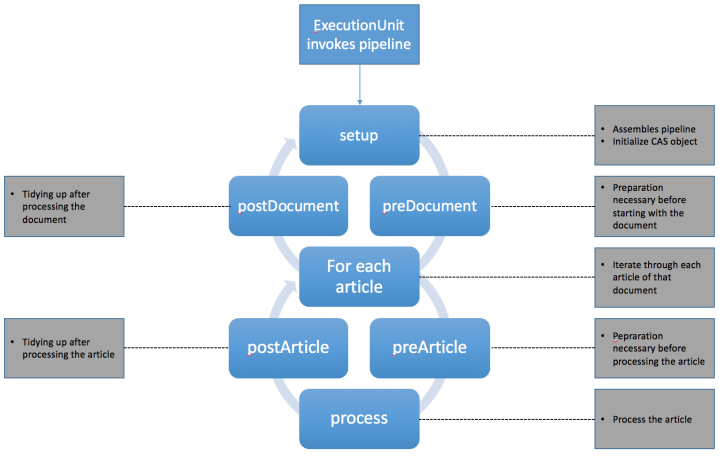


Figure 5.1: Intended life-cycle of the *Pipeline*

Source: Own illustration

GermaNER requires an input format, similar to the CoNNL format. The problem with this format is, that it does not reflect the actual textual representation of contracts within Lexia. Hence, after GermaNER has annotated all the NEs in a given article, all the annotations manually need to be transformed into the native text. Due to that, the pristine text of an article must

be not just attached to the jCas object, but also to different *AnalysisEngines*. According to the typical life-cycle of a pipeline, the setup function is called once for a whole document, which implies that this approach is not suitable for the textitGermaNERPipeline. Figure 5.2 reveals the life-cycle used for the *GermaNERPipeline*.

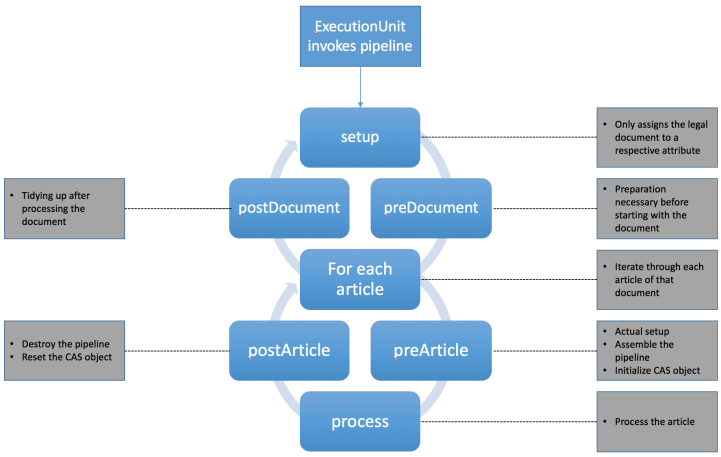


Figure 5.2: Actual life-cycle of the *GermaNERPipeline*

Source: Own illustration

As one can see, the major difference is that the new life-cycle create a pipeline per processed article instead of one pipeline for each document. This is accomplished by using the *setup* function only to store the processed legal document within the pipeline in an attribute. The *preArticle* method performs the actual setup before each article is processed.

Implementation of *GermaNERPipeline*

The previous paragraph introduced the differences between the intended life-cycle by the base class *Pipeline* and the one represented by *GermaNERPipeline*. Due to that reasoning, the implementation of the *GermaNERPipeline* differs from the designated *Pipeline*.

Listing 5.6: Excerpt of *GermaNERPipeline*

```
1  public class GermaNERPipeline extends Pipeline {
2      private LegalDocument document;
3      @Override
4      public void setup(LegalDocument legalDocument, String[] rutaScripts) throws
           ResourceInitializationException, IOException, InvalidXMLException,
           CASException {
5          // No setup of pipeline and cas object necessary here, because we need it for
               each article
6          document = legalDocument;
7      }
8      @Override
9      public AnalysisEngine assemblePipeline(LegalDocument document, String[]
           rutaScripts) throws ResourceInitializationException, IOException,
           InvalidXMLException {
10         return null;
11     }
12     // . . .
13 }
```

The *setup* function of *GermaNERPipeline* only assigns the current contract to the new attribute *document* as shown in Listing 5.6. Furthermore the overriden *assemblePipeline(LegalDocument document, String[] rutaScripts)* is not used and thus, returns null. The suggested *initCas(LegalDocument document, String[] rutaScripts)* is not used either and consequently not overriden. As already described, *preArticle* is responsible for the setup.

Listing 5.7: The method *preArticle*

```
1 @Override
2 public void preArticle(Article article, String text) {
3     try {
4         pipe = assemblePipeline(document, null, text);
5         initCas(document, null, text);
6     }
7     // Catch various exceptions
8 }
```

This setup is shown in Listing 5.7. The new implementation of *assemblePipeline*, which is used by *preArticle* is revealed in the next listing (Listing 5.8).

Listing 5.8: The new *assemblePipeline* method

```
1  public AnalysisEngine assemblePipeline(LegalDocument document, String[] rutaScripts,
       String text) throws ResourceInitializationException, IOException,
       InvalidXMLException {
2      // Create analysis engines for preparation of data
3      AnalysisEngineDescription conllSegmenterDesc =
           createEngineDescription(CoNLLSegmenter.class);
4      AnalysisEngineDescription nerReaderDesc =
           createEngineDescription(NERReader.class, NERReader.DATA_ZIP_FILE, null);
5      AnalysisEngineDescription germaNERPrep =
           createEngineDescription(conllSegmenterDesc, nerReaderDesc);
6
7      // Create analysis engine for GermaNER
8      File modelDirectory = new File("semanticAnalysis/germaNER");
9      modelDirectory.mkdirs();
10     if (!new File(modelDirectory, "model.jar").exists()) {
11         IOUtils.copyLarge(Play.application().resourceAsStream("model/model.jar"),
12             new FileOutputStream(new File(modelDirectory, "model.jar")));
13     }
14     if (!new File(modelDirectory, "MANIFEST.MF").exists()) {
15         IOUtils.copyLarge(Play.application().
16             resourceAsStream("model/MANIFEST.MF"),
17             new FileOutputStream(new File(modelDirectory, "MANIFEST.MF")));
18     }
19     if (!new File(modelDirectory, "feature.xml").exists()) {
20         IOUtils.copyLarge(Play.application().resourceAsStream("feature/feature.xml"),
21             new FileOutputStream(new File(modelDirectory, "feature.xml")));
22     }
23
24     AnalysisEngineDescription germaNERDesc =
           createEngineDescription(NERAnnotator.class,
           GenericJarClassifierFactory.PARAM_CLASSIFIER_JAR_PATH,
           modelDirectory.getAbsolutePath() + "/model.jar",
           NERAnnotator.PARAM_FEATURE_EXTRACTION_FILE,
           modelDirectory.getAbsolutePath() + "/feature.xml",
           NERAnnotator.FEATURE_FILE, modelDirectory.getAbsolutePath());
25     // Create engines to transform to lexia typesystem
26     // . . .
27 }
```

AnalysisEngines for the data preparation are defined in the lines 7 to 9. The *CoNLLSegmenter* is responsible for the transformation of the article's content

into CoNLL format. This is done by utilizing simple regex rules (please refer to Appendix B.1 for the detailed rules). The next pipeline component has to be the *NERReader*. This class is required by GermaNER and also used in its example terminal application. It parses each token in the CoNNL format and applies already different features to these, such as the freebase list. Nonetheless, due to the usage of the Java Play Framework, the instantiation of static files of the *NERReader* had to be changed. Line 9 merges these two engines (*NERReader* and *CoNNLSegmenter*) into one *AnalysisEngineDescrption germaNERPrep*. The next step is the definition of the actual GermaNER analysis engine. This is done in the lines 12 to 30. After defining the location of the GermaNER components, the model is loaded as well as the feature description. Having these resources loaded, the *AnalysisEngineDescription germaNERDesc* can be instantiated. The next step of the pipeline is the transformation of the found NEs into Lexia's type system. Before the way this is done can be described, it must be elaborated a bit on Lexia's typesystem.

Lexia incorporates two type systems in terms of CAS. The first one is about NLP types in general and based on DKPro Core[36], while Lexia extends this type system. For types specific for the legal domain an own type system, which imports the one provided by DKPro Core, has been introduced as well. This type system is extended during this work by the following types: *Person*, *Organisation*, *Location*, *Other*, *Date*, *MoneyValue*. Types for references already existed. Dates, monetary values and references however are not detected by means of GermaNER. Simple regex patterns are used to extract these types. The remaining code of *assemblePipeline* defines the analysis engine to transform the annotated article in CoNNL format into the lexia typesystem. This is accomplished by utilizing regex patterns. For each of the types a pattern class as well as an annotator exists. In place of all these pattern classes, the *PersonPatterns* class is shown in Listing 5.9.

[36] `https://dkpro.github.io/dkpro-core/`

Listing 5.9: The *PersonPatterns* class

```
1  public class PersonPatterns {
2      private static List<String> personPattern = new ArrayList<>();
3      private static List<NamedEntityDiff> personPatternWithContext = new
           ArrayList<>();
4
5      public static String getPersonPattern() {
6          return "(" + String.join("|", personPattern) + ")";
7      }
8      public static List<NamedEntityDiff> getPersonPatternWithContext() {
9          return personPatternWithContext;
10     }
11     public static void addPersonPattern(String pattern) {
12         personPattern.add(pattern);
13     }
14     public static void addPersonPatternWithContext(NamedEntityDiff diff) {
15         if( diff .getNamedEntity().trim().length() > 0) {
16             personPatternWithContext.add(diff);
17         }
18     }
19     public static void clearPattern() {
20         personPattern.clear();
21         personPatternWithContext.clear();
22     }
23 }
```

Such a pattern consists of two attributes. The first attribute *personPattern* holds simply patterns in form of strings. The attribute *personPatternWithContext* bears a list of *NamedEntityDiffs*. For the implementation of the GermaNER pipeline, only *personPattern* is of interest. The attribute *personPatternWithContext* along with *NamedEntityDiff* is described in Section 5.1.4. Representatively for all annotator classes, an excerpt of *PersonAnnotator* is depicted in Listing 5.10.

Listing 5.10: Excerpt of *PersonAnnotator* class

```
1  public class PersonAnnotator extends SegmenterBase {
2      public static final String VIEW = "lexiaTypeView";
3      @ConfigurationParameter(name = VIEW, mandatory = false)
4      private String view = null;
5
6      @Override
7      protected void process(JCas aJCas, String text, int zoneBegin) throws
           AnalysisEngineProcessException {
8
9          if (view != null) {
10             try {
11                 aJCas = aJCas.getView(view);
12                 text = aJCas.getDocumentText();
13             }
14             catch(CASException e) {
15                 e.printStackTrace();
16             }
17         }
18         List<Annotation> foundAnnotations = new ArrayList<>();
19
20         String personPatternString = PersonPatterns.getPersonPattern();
21         Pattern personPattern = Pattern.compile(personPatternString);
22         Matcher m = personPattern.matcher(text);
23         while (m.find()) {
24             if (m.start() == m.end() ||
                   checkAnnotationAlreadyFound(foundAnnotations, m.start()) == true)
                   continue;
25             Annotation personAnnotation = createPersonAnnotation(aJCas, m.start(),
                   m.end());
26             foundAnnotations.add(personAnnotation);
27         }
28     }
29 }
```

The configuration parameter of line 2 to 4 is necessary, to provide the *PersonAnnotator* with the proper CAS view, containing the native unannotated text of the article being processed. However, for other purposes the class may be used without that configuration parameter and thus, line 9 to 17 performs a check whether this view is supposed to be used or not. The actual annotation process is accomplished from line 20 to 27. The respective pattern is extracted from the *PersonPatterns* and assigned to *personPatternString*. A *personPat-*

tern is defined by means of the extracted *personPatternString*. Now a *Matcher* can proceed through the text in order to find each occurrence of any pattern within the *personPattern*. Each appearance is then turned into an annotation of type *Person* in line 25, in case it was not already persisted. More precisely, the found NEs of type *Person* by GermaNER shape the used *Pattern* and thus, this class turns each recognized NE into an annotation of the respective lexia type. As already mentioned above, three types are recognized by utilizing only regex. The same classes exist for these types, the only difference is that the pattern classes contain static pre-defined patterns.

Listing 5.11: Creation of analysis engines for the type system transformation

```
1 AnalysisEngineDescription neTransformerDesc =
      createEngineDescription(NETransformer.class, NETransformer.FULL_TEXT, text);
2 AnalysisEngineDescription personDesc = createEngineDescription(PersonAnnotator.class,
      PersonAnnotator.VIEW, NETransformer.LEXIA_TYPES_VIEW);
```

The transformation requires several analysis engines. Listing 5.11 depicts two of these. The first line creates the *NETransformer*, which is necessary to populated the pattern classes with the relevant regex patterns (Appendix B.2). Line 2 creates already the *PersonAnnotator*, while all other annotators follow the same principle. After all *AnalysisEngines* has been created, the final pipeline can be constructed. Now the new *initCas* function can be called within *preArticle*. The following listing (Listing 5.12) includes the source code of that function.

Listing 5.12: The method *initCas* of *GermaNERPipeline*

```
1 protected void initCas(LegalDocument legalDocument, String[] rutaScripts, String text)
      throws ResourceInitializationException, IOException, InvalidXMLException,
      CASException {
2         jCas = UimaUtil.produceJCas();
3         jCas.createView(CoNLLSegmenter.FULL_VIEW);
4         jCas.getView(CoNLLSegmenter.FULL_VIEW).setDocumentText(text);
5         jCas.setDocumentLanguage(legalDocument.getLanguage());
6     }
```

After the *jCas* object is created by means of *UimaUtil*, a specific view is created. The *CoNNLSegmenter* formats the article text into the CoNNL format. This is done in the default view of the CAS object, because the *NERReader* expects its input in that default view. Hence, the *CoNLLSegmenter* gets its

own input, the native text, in an extra view. This is done in the lines 3 and 4. The final line only sets the language of the document to German. With that said, the setup is done and the actual pipeline of GermaNER can be invoked. The *process* function is supposed to do so (Listing 5.13).

Listing 5.13: The method *process* of *GermaNERPipeline*

```
1  @Override
2      public PipelineResult process(Article article, String text) throws
           AnalysisEngineProcessException {
3          PipelineUtil.setAnnotationWhiteList(Lists.newArrayList("informationExtraction.
4          lexiaTypes.basicEntities.Location"
5          // All other types are added here
6          ));
7
8          pipe.process(jCas);
9          try {
10             jCas = jCas.getView(NETransformer.LEXIA_TYPES_VIEW);
11         }
12         catch(Exception e) {
13             e.printStackTrace();
14         }
15         PipelineUtil.getAnnotationWhiteList().clear();
16         PipelineResult result = createAnnotationStructures(article);
17         return result;
18     }
```

In the lines 3 to 6, all types being created during the pipeline, are added to a annotation white list. This is necessary, because Lexia is configured to only store annotations being white-listed. This is the case in order to prevent accidental storage of unwanted annotations. The actual pipeline is invoked in line 8. As already mentioned, the annotator classes work on a specific CAS view. This view is extracted in line 10. Afterwards the annotation white list is cleared before the actual annotation structures are created.

Accessibility from the REST API

After explaining the implementation of the concrete *GermaNERPipeline*, the remainder of this section discusses how this pipeline is made available to the frontend via the REST API. The respective endpoint is */api/semanticanalysis/pipeline*. This endpoint routes the request to the *PipelineController*'s

function *runNERPipeline*. The method parses the JSON body where it extracts the documents to be processed as well as the pipeline. It figures out the type of each document and consolidates all documents in a list of *LegalDocuments*. Afterwards a *DocumentCorpus* is created by utilizing this list. The corpus is responsible to process itself with the pipeline. For that reason, the function *processCorpusWithPipeline* iterates over each document within the corpus and utilizes the *PipelineExecutor* in order to perform the NER task.

Listing 5.14: Excerpt of the method *processCorpusWithPipeline* of *DocumentCorpus*

```
1  public JSONArray processCorpusWithPipeline(String pipeline) {
2          int pipeNr;
3          // Retrieve the corresponding pipeline number
4
5          JSONArray returnObject = new JSONArray();
6          while(!corpusElements.isEmpty()) {
7              LegalDocument document = corpusElements.get(0);
8              Predicate<Article> articlePredicate = x -> !x.filtersApply(-1, -1, "");
9              try {
10                 document = new PipelineExecutor().runPipeline(document, null, pipeNr,
                       articlePredicate, true);
11                 // Create the return object for this document
12                 corpusElements.remove(0);
13             }
14             catch(Exception e) {
15                 e.printStackTrace();
16                 return null;
17             }
18         }
19         return returnObject;
20     }
```

The pipeline invocation is shown in line 10 of the Listing 5.14. Before that, the function figures out the pipeline number of the requested pipeline, which is required by the *PipelineExecutor*. The remainder of the method assembles the JSON response, which is just forwarded by the *PipelineController* to the UI. The structure of such a response is depicted in Listing 5.15.

Listing 5.15: Example response for the */api/semanticanalysis/pipeline* request

```
[
  {
    "legalDocumentId" :"1",
    " articleEntities " : [
      {
        " articleId " : "11",
        " entities "  : [
          {
            "coveredText"   : "TUM",
            "legalDocument" :"1",
            " article "     : "11",
            "begin"         : "10",
            "end"           : "12",
            "types"         : "Organisation",
            "annotationId"  : "111"
          }
        ]
      }
    ]
  }
]
```

The response includes one entry per processed legal document. Hereby each entry contains the id of the document (*legalDocumentId*)as well as another attribute *articleEntities*. The attribute compromises one entry in its array for each article. Again, each entry reveals the *articleId* along with an attribute *entities*. That attribute has one entry per created annotation.

5.1.3.2 DBpedia Spotlight

NFR06 from Chapter 4.4.2 requires the integration of DBpedia Spotlight as one NER technology. DBpedia Spotlight has been already introduced in Chapter 4.1.2.2. This section deals with the integration into the prototypical implementation of this work.

Lifecycle of *DBPediaPipeline*

One important aspect of software engineering is consistency within an application or component. Hence, the life-cycle of the *DBPediaPipeline* orients oneself by the one of the *GermaNERPipeline* (see first Paragraph of 5.1.3.1). In fact, its life-cycle mirrors the life-cycle form the *GermaNERPipeline* (cf. Figure 5.2).

Implementation of *DBPediaPipeline*

In contrast to the life-cycle of this pipeline, the implementation differs quite a bit from the pipeline utilizing GermaNER. While the *setup* function of *DBPediaPipeline* as well as *preArticle* is exactly the same, *assemblePipeline* and *initCas* provide their own distinct implementations.

Listing 5.16: Excerpt of *assemblePipeline* from *DBPediaPipeline*

```
1  public AnalysisEngine assemblePipeline(LegalDocument document, String[] rutaScripts,
       String text) throws ResourceInitializationException, IOException,
       InvalidXMLException {
2          AnalysisEngineDescription spotlightTagger =
               createEngineDescription(SpotlightAnnotator.class,
               SpotlightAnnotator.PARAM_CONFIDENCE, 0.5f,
               SpotlightAnnotator.PARAM_ENDPOINT,
               "http://model.dbpedia-spotlight.org/de/annotate");
3
4          // Create engines to transform to lexia typesystem
5          AnalysisEngineDescription neTransformerDesc =
               createEngineDescription(DBPediaNETransformer.class,
               DBPediaNETransformer.FULL_TEXT, text);
6          AnalysisEngineDescription personDesc =
               createEngineDescription(PersonAnnotator.class, PersonAnnotator.VIEW,
               NETransformer.LEXIA_TYPES_VIEW);
7          // Create the other AnalysisEngines
8          AnalysisEngineDescription transformationDesc =
               createEngineDescription(neTransformerDesc, personDesc, organisationDesc,
               locationDesc, otherDesc, dateDesc, moneyDesc);
9
10         // Put together final pipeline
11         AnalysisEngineDescription pipeDesc = createEngineDescription(spotlightTagger,
               transformationDesc);
12         AnalysisEngine pipe = createEngine(pipeDesc);
13
14         return pipe;
15     }
```

The source code in Listing 5.16 shows an excerpt of the source code of *assemblePipeline*. Line 2 already defines the *AnalysisEngine spotlightTagger*. The class *SpotlightAnnotator* which is used in that step, is provided by DBpedia Spotlight. As already described in Chapter 4.1.2.2, the API offers various configuration parameters. Those parameters are available in the interface of *SpotlightAnnotator*, too. A confidence of .5 has been chosen. This specific confidence was picked, based of several tests. Other than the actual API endpoint, no configuration is provided. For the sake of consistency, obviously the same type system as used by the *GermaNERPipeline* shall be utilized for the final annotations. That is the reason why line 5 to 8 create the *AnalysisEngines* for this transformation process, similar to *GermaNERPipeline*. In fact, the ap-

proach is the same, incorporating the equal amount of pattern and annotator classes. However, a specific class is required for populating the patterns, that is *DBPediaNETransformer* (please have a look at Appendix B.3 for further information). This is justified by the fact that *SpotlightAnnotator* uses a different type-system than *NERAnnotator* from GermaNER. The final pipeline is assembled in line 12. According to the life-cycle, the *setup* function calls the method *initCas*, once the pipeline is created. Only the article text is assigned to the *jCas* object, created via *UimaUtil*, in that *initCas* function. No additional CAS views are required for this pipeline, other than the resulting lexia types view, because the *SpotlightAnnotator* is able to process the native text. The *process* method, invoking the actual pipeline, again is totally similar to the one of *GermaNERPipeline*.

Accessibility from the REST API

As described in Chapter 4.5.5, all NER pipelines use the same API endpoint. The only difference between the pipelines is the different pipeline name, which is sent in the request body. As a result, no additional description of the accessibility is required at this point. The path from the user's request to the pipeline is the exact same as it has been shown for the pipeline incorporating GermaNER (cf. third Paragraph in Section 5.1.3.1).

5.1.3.3 Templated

According to NFR07 in Chapter 4.4.2, templated NER is the final approach to NER in the course of this work. The implementation of that approach is discussed in this section.

Mechanis behind templated NER

The technique behind templated NER has not been discussed yet, this is done in this paragraph. The main idea of templated NER is, as elaborated on in Chapter 4.1.4, that each legal contract is based on a template. When this template is available, we can use it to extract the NEs which need to be filled when creating a specific contract instance.

Listing 5.17: Example sentence from a template

Vertragsgegenstand ist die Lieferung von insgesamt −−Verkaufsprodukt.Menge−− Stück −−Verkaufsprodukt.Name−− des Herstellers −−Verkaufsprodukt.Hersteller−−.

A possible example of a sentence from such a template is shown in Listing 5.17. For the placeholders, a concept has been used, where two dashes followed by some word ensued by another two dashes indicate a NE. With other words, the following regex highlights a placeholder *"(–)*.(–)"*. During the contract creation process, such a template may be filled as follows.

Listing 5.18: Example sentence from a instantiated template

Vertragsgegenstand ist die Lieferung von insgesamt 12 Stück MacBook Pros des Herstellers Apple.

Listing 5.18 depicts an instantiated template. The goal of a templated NER approach, is to extract the three NEs: (1) *12*, (2) *MacBook Pros*, and (3) *Apple*. By comparing the template and the instance, it is exposed that those three NEs are the only difference between the two sentence. That is already the concept behind templated NER, which is implemented in the course of this study. In order to implement it, Google's *Diff-Match-Patch*[37] (DMP) algorithm is utilized. The algorithm is based on Myer's diff algorithm[98]. When executing the algorithm, only pairs of differences augmented by the diff-option (equal, insert, and delete) are returned. The *DiffController* utilizing this algorithm, only return the pair of strings, containing the placeholder value along with the actual value of the found NE.

Furthermore, the *DiffController* offers a function *findTemplateToArticleInstance* which returns the appropriate template to a given article.

[37] `http://code.google.com/archive/p/google-diff-match-patch`

Listing 5.19: Method *findTemplateToArticleInstance* of the *DiffController*

```
1  public static String findTemplateToArticleInstance(String id, String instanceContent) {
2
3          List<String> templateCandidates = retrieveMoreArticlesLikeThis(id);
4
5          List<String> instanceList =
               Arrays.asList(instanceContent.split(TOKENIZATION_REGEX));
6
7          String mostPromisingTemplate = "";
8          int mostPromisingTemplateLCS = -1;
9
10         for (String templateCandidate : templateCandidates) {
11             List<String> templateList =
                   Arrays.asList(HtmlUtil.convertToPlaintext(templateCandidate).split("
                   "));
12
13             List<String> result =
                   org.apache.commons.collections4.ListUtils.longestCommonSubsequence(templateList,
                   instanceList);
14
15             Logger.info("LCS Length: " + result.size());
16
17             if (result.size() > mostPromisingTemplateLCS) {
18                 mostPromisingTemplateLCS = result.size();
19                 mostPromisingTemplate = templateCandidate;
20             }
21         }
22
23         return mostPromisingTemplate;
24     }
```

Listing 5.19 includes the code of this function. Line 3 of the source code retrieves initially all possible template candidates. This is done by incorporating Elasticsearch's *More Like This* (MLT) query. Each returned article is then searched by the placeholder pattern. If it contains at least one placeholder pattern, it is a potential template candidate. Before returning the final candidate list to *findTemplateToArticleInstance*, the candidates are ordered by the MLT score. Afterwards for each template candidate, the longest common subsequence with the article instance is calculated. Finally, the most promising template is returned.

Lifecycle of *TemplatedNERPipeline*

Just like *DBPediaPipeline*, also *TemplatedNERPipeline* is based on the exact same life-cycle as *GermaNERPipeline* (cf. Figure 5.2).

Implementation of *TemplatedNERPipeline*

In comparison to *DBPediaPipeline*, the implementation of *TemplatedNERPipeline* even differs greater from *GermaNERPipeline*. The *setup* function, the method to initialize the CAS object *initCas* as well as *preArticle* remain the same, while *assemblePipeline* is quite a bit different, as illustrated in Listing 5.20.

Listing 5.20: The method *assemblePipeline* of *TemplatedNERPipeline*

```
1  public AnalysisEngine assemblePipeline(LegalDocument document, String[] rutaScripts,
       String text) throws ResourceInitializationException, IOException,
       InvalidXMLException {
2          AnalysisEngineDescription personDesc =
               createEngineDescription(PersonNamedEntityDiffAnnotator.class,
               PersonAnnotator.IGNORE_TAGS, true);
3          AnalysisEngineDescription organisationDesc =
               createEngineDescription(OrganisationNamedEntityDiffAnnotator.class,
               OrganisationAnnotator.IGNORE_TAGS, true);
4          AnalysisEngineDescription locationDesc =
               createEngineDescription(LocationNamedEntityDiffAnnotator.class,
               LocationAnnotator.IGNORE_TAGS, true);
5          AnalysisEngineDescription otherDesc =
               createEngineDescription(OtherNamedEntityDiffAnnotator.class,
               OtherAnnotator.IGNORE_TAGS, true);
6          AnalysisEngineDescription transformationDesc =
               createEngineDescription(personDesc, organisationDesc, locationDesc,
               otherDesc);
7
8          // Put together final  pipeline
9          AnalysisEngine pipe = createEngine(transformationDesc);
10
11         return pipe;
12     }
```

Even though the function itself only creates the *AnalysisEngines* to annotate the article text based on patterns, different annotator classes are used. This is necessary due to the mechanics behind templated NER. As already explained in the previous paragraph (first Paragraph of 5.1.3.3), the templated NER is based on Google's DMP algorithm and only returns a list of differences, whereas a difference consists of the two different text phrases. In order to create annotations for these differences (representing NEs), all NEs could be populated in the respective patterns. However this brings along a big problem. Imagine a purchase agreement. Obviously such an agreement includes the number of commercial products. If this number would be *1*, and this is populated into the patterns, in the process of creating the annotations each occurence of the number *1* is annotated. It is obvious that this would not be the intended behavior of such a system. Due to that reasoning, specific classes are created for the annotation of NEs recognized via templated NER. The class *PersonNamedEntityDiffAnnotator* is used to explain the functionality of those annotators, while each type has its own annotator.

Listing 5.21: Excerpt of *PersonNamedEntityDiffAnnotator*

```
1  public class PersonNamedEntityDiffAnnotator extends SegmenterBase {
2
3      public static final String VIEW = "lexiaTypeView";
4      @ConfigurationParameter(name = VIEW, mandatory = false)
5      private String view = null;
6
7      public static final String IGNORE_TAGS = "ignoreTags";
8      @ConfigurationParameter(name = IGNORE_TAGS, mandatory = false)
9      private Boolean ignoreTags = false;
10
11     @Override
12     protected void process(JCas aJCas, String text, int zoneBegin) throws
           AnalysisEngineProcessException {
13         // Implementation
14     }
15
16     private boolean checkAnnotationAlreadyFound(List<Annotation>
           foundAnnotations, int start) {
17         // Implementation
18     }
19
20     protected Person createPersonAnnotation(JCas aJCas, final int aBegin, final int
           aEnd, String type) {
21         // Implementation
22     }
23 }
```

Listing 5.21 shows an excerpt of the *PersonNamedEntityDiffAnnotator*. The structure of the class conforms to the one of the simple annotators, used by the *GermaNERPipeline* and *DBPediaPipeline*. The difference is the implementation of the *process* method (Listing 5.22).

Listing 5.22: Excerpt of the *process* method from *PersonNamedEntityDiffAnnotator*

```
1  @Override
2  protected void process(JCas aJCas, String text, int zoneBegin) throws
       AnalysisEngineProcessException {
3      // Check which view to use . . .
4
5      List<Annotation> foundAnnotations = new ArrayList<>();
6      List<NamedEntityDiff> entities = PersonPatterns.getPersonPatternWithContext();
7      for (NamedEntityDiff entity : entities ) {
8          Pattern personPattern = Pattern.compile(entity.getNamedEntity());
9          Matcher m = personPattern.matcher(text);
10         while(m.find()) {
11             if (m.start() == m.end() ||
                   checkAnnotationAlreadyFound(foundAnnotations, m.start()) == true)
                   continue;
12
13             Boolean ignoreAnnotation = false;
14             // If ignoreTags is true, ignore those . . .
15
16             String foundPattern = text.substring(m.start(), m.end());
17             ignoreAnnotation = true;
18             Pattern contextPattern = Pattern.compile(entity.getContext());
19             Matcher contextMatcher = contextPattern.matcher(text);
20             while(contextMatcher.find()) {
21                 if (contextMatcher.start() <= m.start() && contextMatcher.end() >=
                       m.end()) {
22                     ignoreAnnotation = false;
23                 }
24             }
25             if (ignoreAnnotation) continue;
26
27             Annotation personAnnotation = createPersonAnnotation(aJCas, m.start(),
                   m.end(), entity.getType());
28             foundAnnotations.add(personAnnotation);
29         }
30     }
31 }
```

After checking which view to use for the processing (similar to the approach in *PersonAnnotator* and not shown in the listing above), the patterns are extracted. Hereby, also the class *PersonPatterns* is used, but this time the func-

tion *getPersonPatternWithContext* is claimed. The population of the respective attribute of *PersonPatterns*, *personPatternWithContext* is described further down, when talking about the *process* method of *TemplatedNERPipeline*. For each NE returned by *PersonPatterns*, the text of the NE itself is used to create a pattern. Now the whole article text is searched for this pattern. Hereby each occurrence is first checked whether it has been already found or not. Afterwards, it is verified that the NE is not within a HTML tag, in case the *ignoreTags* attribute is set to true. The next step is the most important one and at the same time the biggest difference to the other annotator classes, introduced so far. The context of the found NE is extracted and used to build a pattern as well. Then the whole text is searched for this specific context. Once the context is found, it is reviewed whether the found NE is within that context in the native text. Just if that is the case, the annotation is created.

Once the pipeline is assembled, it can be invoked by the *process* function, which differs a lot from the previous ones. Listing 5.23 depicts the implementation of this method.

Listing 5.23: Excerpt of the *process* method from *TemplatedNERPipeline*

```
1  public PipelineResult process(Article article, String text) throws
       AnalysisEngineProcessException {
2
3          String instanceContent = text;
4          String template = DiffController.findTemplateToArticleInstance(article.getID(),
               text);
5          ArrayList<Pair<String, String>> templatedNERResult =
               DiffController.determineDifferences(instanceContent, template, false);
6
7          // Clear patterns
8
9          for(Pair<String, String> entity : templatedNERResult) {
10             // Determine the context of each found entity . . .
11
12             NamedEntityDiff diff = new NamedEntityDiff(entity.getKey(),
                   entity.getValue(), context, startOffset, endOffset);
13
14                 if(diff.getType().contains("Location")) {
15                     LocationPatterns.addLocationPatternWithContext(diff);
16                 }
17                 else if(diff.getType().contains("Organisation") ||
                       diff.getType().contains("Organization")) {
18                     OrganisationPatterns.addOrganisationPatternWithContext(diff);
19                 }
20                 else if(diff.getType().contains("Person")) {
21                     PersonPatterns.addPersonPatternWithContext(diff);
22                 }
23                 else {
24                     OtherPatterns.addOtherPatternWithContext(diff);
25                 }
26         }
27
28         PipelineUtil.setAnnotationWhiteList(Lists.newArrayList("informationExtraction.
29             lexiaTypes.basicEntities.Location",
30             // Other types added
31             ));
32         pipe.process(jCas);
33         PipelineUtil.getAnnotationWhiteList().clear();
34         PipelineResult result = createAnnotationStructures(article);
35         return result;
36     }
```

The first step of *process* is to determine the suitable template for the current article. For that reason, *findTemplateToArticleInstance* of the *DiffController* is utilized. This function is based on Elasticsearch's MLT query. Once the template is ascertained, the actual differentiation can be performed. For this purpose, *DiffController*'s *determineDifferences* is invoked. The resulting list of pairs of strings is stored in *templatedNERResult*. As already mentioned, before each NE can be populated via patterns, the context of a NE need to be identified. A window of -10 to +10 characters is used as the context. The source code of this task is not revealed in Listing 5.23, because it is way to complex to be inserted in such a thesis. This is due to the various exceptions, which need to be regarded, such as having two NEs within one context. After the context is determined, line 12 creates the *NamedEntityDiff* object which holds next to the content of a NE, the type of a NE, the actual value, as well as start offset and end offset. Depending on the type, the NE can be populated via the existing pattern classes. The remainder of the *process* method executes the pipeline and returns the result.

Accessibility from the REST API

Exactly as for the *DBPediaPipeline*, the same route utilized by *GermaNER-Pipeline* is used for the *TemplatedNERPipeline*, too. This implies that the structure of the JSON response remains also untouched (cf. Listing 5.15 in Section 5.1.3.1).

5.1.4 NED

NED based on a templated approach was already briefly introduced in Chapter 4.2.3. This section deals with the actual implementation of such an approach. In order to perform the disambiguation process, models of contracts are necessary. Hence, Section 5.1.4.1 discusses the semantic models, which are used for that purpose. Then, the actual disambiguation component is elaborated in Section 5.1.4.2.

5.1.4.1 Semantic Models

The elicitation of NFR07 in Chapter 4.4.2 asks for the utilization of the modeling domain from Oppmann's[106] implementation concerning a modeling environment. In the course of this work, the class *Model*, representing semantic models of legal documents, has been extended to suit the needs for this work as well.

Listing 5.24: Excerpt of the *Model* class

```
1  public class Model extends Entity {
2
3      private String scWorkspaceId;
4      private String id;
5      private String  title ;
6      private Date createdAt;
7      private Date updatedAt;
8      private String jsonModelDefinition;
9      private String jsonDocumentReferences;
10     private Boolean contractModel;
11
12     public final static String SC_TYPE = "Model";
13     public final static String SC_TYPE_PLURAL = "Models";
14
15     // Constructor, as well as Getter and Setter
16
17     // Methods to store, update, delete and retrieve Models
18 }
```

An excerpt of the *Model* class is shown in Listing 5.24. Next to the code shown in the listing above, the class includes a constructor, getter and setter methods, and various functions to store, update and retrieve models. The *Model* offers way more functionality than required in this study. That is the reason, why a new attribute *contractModel* is introduced. This attribute is set to true, for all models, representing a contract and thus being used in the disambiguation process. Furthermore, the existing attributes *id*, *title*, *createdAt*, and *updatedAt* are self-explanatory. As already discussed in Chapter 4.4.2 (cf. NFR08 and NFR09), both, Elasticsearch and SocioCortex shall be used for the storage of models. This was already supported by Oppmann's implementation. The *Model* as it is described here, is stored in the Elasticsearch database. Hereby, the *jsonModelDefinition* contains the actual semantic definition of a model

along with all the instances of such a model. However, in the *Model* and thus in Elasticsearch, it is only depicted as a JSON object. A model is created and updated in the UI by means of JointJS (see Section 5.2 for detailed information). The resulting definition is represented by the *jsonModelDefiniton*. This JSON representation is then mapped to entities of SocioCortex (as described in Chapter 4.5.3) and stored there. The *scWorkspaceId* stores the id of the respective SocioCortex workspace. In order to store and retrieve the models in the SocioCortex, the *SocioCortexControllerV2* created by Oppmann is utilized.

5.1.4.2 Disambiguation Component

Unfortunately, the disambiguation component is not implemented as an Apache UIMA component and thus does not reflect a pipeline. The way the communication within an Apache UIMA pipeline is handled, is not beneficial for the purpose of linking NEs from a legal contract to a semantic model. This is because these pipelines are designed to perform annotation tasks. To be more precise, a CAS object only stores information about different annotations concerning its text, but no information regarding external entities, such as the *Model*. Nonetheless, the extensibility demanded by NFR03 is respected in this implementation.

5.1.4.2.1 Implementation

A class only to perform linkings by means of different approaches is created, that is *EntityLinker*. Yet it only covers one method, which is used to link the NEs of a contract, which were extracted via templated NER, to a semantic contract model. An excerpt of this function is shown in Listing 5.25.

Listing 5.25: Excerpt of the method *linkModelToContractTemplated*

```
1  public static Model linkModelToContractTemplated(Model model, JsonNode document,
       JsonNode entities) {
2      try {
3          JSONObject modelJson = new
               JSONObject(model.getJsonModelDefinition());
4          JSONArray cells = modelJson.getJSONArray("cells");
5
6          for(int i = 0; i < cells.length(); ++i) {
7              JSONObject payload =
                   cells.getJSONObject(i).getJSONObject("payload");
8              String typeTitle = payload.getString("title");
9              JSONArray attributes = payload.getJSONArray("attributes");
10             JSONObject instance = new JSONObject();
11             for(int j = 0; j < attributes.length(); ++j) {
12                 JSONObject attribute = attributes.getJSONObject(j);
13                 String attributeName = attribute.getString("name");
14                 for(JsonNode entity : entities) {
15                     String[] nodes =
                           entity.get("entityType").toString().replaceAll("--",
                           "").replaceAll("\"", "").split("\\.");
16                     if(typeTitle.equals(nodes[0]) &&
                           attributeName.equals(nodes[1])) {
17                         instance.put(attributeName,
                               entity.get("annotationId").toString().replace("\"",
                               ""));
18                     }
19                 }
20             }
21
22             instance.put("SC_NAME", typeTitle + "_" +
                   document.get("id").toString().replaceAll("\"", ""));
23             // Check whether a link for that specific Model and Contract already
                   exist and if so, replace the link
24         }
25
26         // Transform back to JSON representation and store
27     }
28     catch(Exception e) {
29         e.printStackTrace();
30         return null;
31     }
32     return model;
33 }
```

The first part of the method *linkModelToContractTemplated*, dismissing some trivial code to check whether this contract is already linked to that model, reveals the code for the actual linking. The function retrieves the semantic *Model*, the legal contract represented in JSON, as well as all recognized NEs also in the JSON format. First, the *jsonModelDefiniton* is traversed by *cells*. Each cell represents one type. Second, each type is traversed by *attributes*. That is the place where the disambiguation takes place, because one attribute must be linked to one NE. As already described in Chapter 4.2.3, the key idea of this disambiguation approach is the following. When a contract model is defined, types as well as attributes are required. A conceptual JSON representation of such a model is shown in Listing 5.26.

Listing 5.26: Conceptual example of a model

```
[
    {
        "type" : "Tenant"
        "attributes" : [
            "Firstname",
            "Surname"
        ]
    },
    {
        "type" : "Landlord"
        "attributes" : [
            "Firstname",
            "Surname"
        ]
    }
]
```

In this example, a contract includes a tenant, as well as a landlord. The tenant and the landlord are types. Both of these two types, have two attributes: (1) first name, and (2) surname. Now this just needs to be respected in the creation process of a template.

Listing 5.27: Example of a sentence from a template

```
1 --Landlord.Firstname-- --Landlord.Surname-- rents his Apartment to
    --Tenant.Firstname-- --Tenant.Surname--.
```

A possible sentence from a template is revealed in Listing 5.27. Through the usage of the type and attribute names, defined in the model, the links can be already disambiguated in the template creation process. Now the code from line 14 to 19 of Listing 5.25 only needs to find the NE with the type name corresponding to the current attribute. Once this is done for each attribute, the JSON representation of the model definition must be transformed back for the storage in SocioCortex. This is depicted in Listing 5.28.

Listing 5.28: Excerpt of the method *linkModelToContractTemplated* (continued)

```
1  public static Model linkModelToContractTemplated(Model model, JsonNode document,
       JsonNode entities) {
2          try {
3              // Disambiguate and set the actual links
4
5              JointJSModelConverter converter = new JointJSModelConverter();
6              ObjectMapper mapper = new ObjectMapper();
7              JsonNode jsonModel = mapper.readTree(modelJson.toString());
8
9              model.setJsonDocumentReferences(references.toString());
10             converter.saveInstancesInSocioCortex(jsonModel, model);
11         }
12         catch(Exception e) {
13             e.printStackTrace();
14             return null;
15         }
16         return model;
17     }
```

First, a mapper is used to read the JSON representation and to transform it back to a *JsonNode*. Second, the document references are updated at the *model*, before the *JointJSModelConverter* is utilized to save the instances of the model back to SocioCortex.

5.1.4.2.2 Accessibility from the REST API

In order to make the disambiguation component available to the front-end, a *EntityLinkingController* is implemented. In detail, the function *linkTemplat-*

edEntities is bind to the route */api/semanticanalysis/entitylinking/templated-ner*.

Listing 5.29: Excerpt of the method *linkTemplatedEntities*

```
1  public static Result linkTemplatedEntities() {
2          String linkingApproach =
               request().body().asJson().findValue("approach").asText();
3          String modelId = request().body().asJson().findValue("model").asText();
4          JsonNode contract = request().body().asJson().findValue("document");
5          JsonNode entities = request().body().asJson().findValue("entities");
6          Model contractModel;
7
8          if (linkingApproach == "automatic") {
9              contractModel = Model.determineModelForContract(entities);
10         }
11         else if (modelId != null) {
12             contractModel = Model.loadFromElasticSearch(modelId);
13         }
14
15         contractModel = EntityLinker.linkModelToContractTemplated(contractModel,
               contract, entities);
16
17         JSONArray returnObject = new JSONArray();
18         // Create the JSON response
19
20         catch(Exception e) {
21             e.printStackTrace();
22             return internalServerError(e.toString());
23         }
24
25         return ok(returnObject.toString());
26     }
```

This function is partially shown in Listing 5.29. After extracting the required arguments from the JSON body, the model which shall be linked is determined. The user can either specify the model in the UI, or choose the automatic identification. Therefore the function *determineModelForContract* determines the suitable model, based on the best match in terms of equal names between types and attributes of the model and the types of the recognized NEs. Afterwards the disambiguation is invoked in line 15. Then only the JSON response is built before it is returned to the UI.

5.2 UI

After describing the back-end implementation, the front-end is discussed in this section. Figure 5.3 shows the components of the UI. They are separated in three categories: (1) *Controllers* along with corresponding *views*, (2) *services*, and (3) *directives*. Hereby, the majority of the components are reused from Oppmann[106], even though most of those components were adapted to fit the needs of this thesis. The following sections describe these components in closer detail.

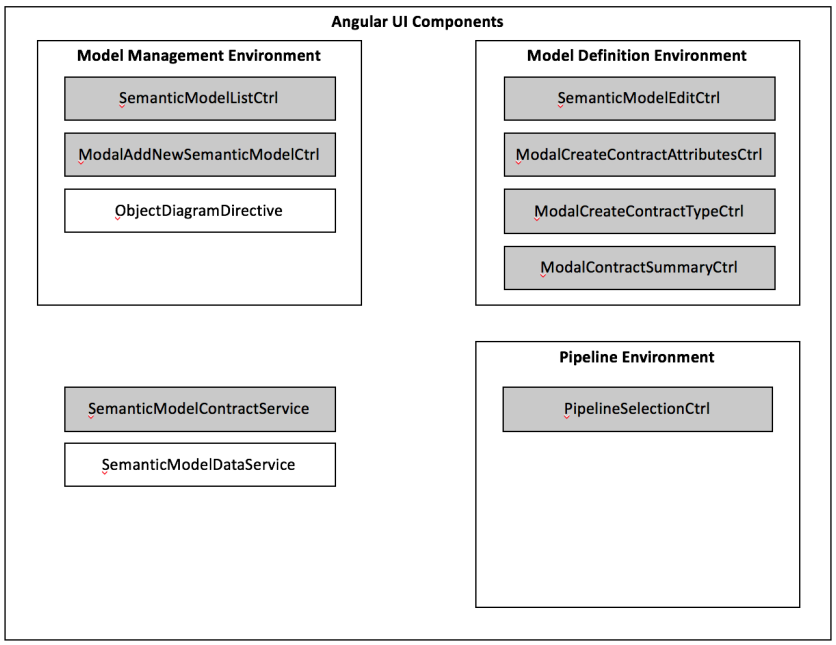


Figure 5.3: Components of the front-end implementation

Source: Own illustration

5.2.1 Model Environment

This section describes the front-end implementation concerning components related to the creation, update, and deletion process of semantic models.

5.2.1.1 Services for the Model Environment

AngularJS services are a nice way to centralize back-end requests in a proper interface. Hence, the front-end utilizes two services, which provide API requests as well as interfaces to functionalities of other libraries.

SemanticModelContractService

As already mentioned in Chapter 4.5.2, the JointJS JavaScript library is used to graphically represent contract models. Furthermore, the library provides the JSON representation of such a model, which is mapped to SocioCortex. Oppman created a custom AngularJS service to encapsulate the necessary JointJS functionality and to expose only a well-defined interface. Since the modeling environment of Oppmann is far more complex than the one used for this work, the class has been modified for the purpose of this thesis. For a detailed description about the implementation, please refer to Oppmann[106, p. 81-82].

SemanticModelDataService

The *SemanticModelDataService* only encapsulates all requests to the back-end component of the modeling component. This allows the different controllers to use a well-defined and centralized interfaces, rather than creating each request manually again.

5.2.1.2 Model Management

The *Model Managment Environment* component provides several AngularJS controllers and views, as well as modal dialogs. This component compromises also a directive. In this section, the different views and their functionality are discussed briefly. Moreover, screenshots are provided for those views.

SemanticModelListCtrl

The *SemanticModelListCtrl* provides a tabular overview over all semantic models stored in Lexia, which relate to contracts. This view serves as an entry point, from which the user can either navigate to the model editor, or import and export existing models. Furthermore, the user can create initially a model as well as delete old models. Figure 5.4 provides a screenshot of this view.

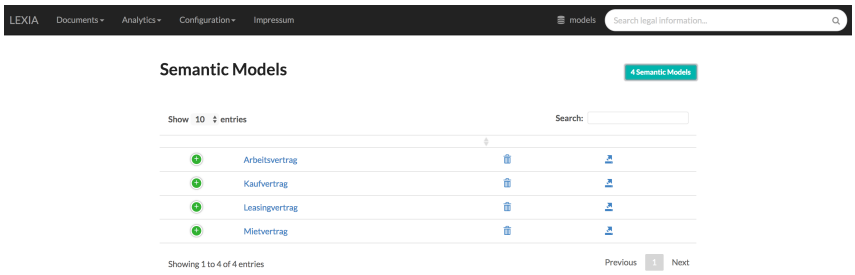


Figure 5.4: Screenshot of the model overview

Source: Own screenshot

ModalAddNewSemanticModelCtrl

This view is presented as a modal view which is used to create new semantic models. It just shows a text box in which the user must enter the name of the model. Then the *ModalAddNewSemanticModelCtrl* creates the model and sends it to the back-end. This view is shown in Figure 5.5.

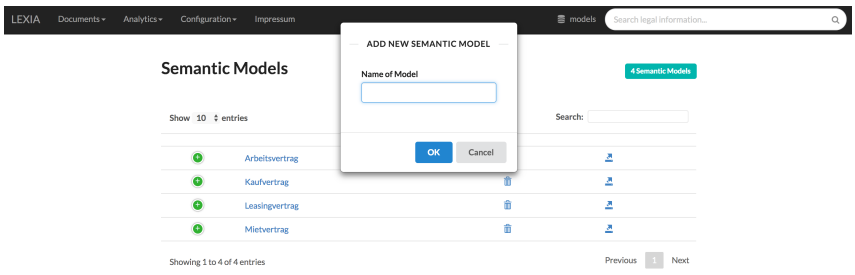


Figure 5.5: Screenshot of the view to create a model

Source: Own screenshot

ObjectDiagramDirective

This directive has been inherited by Oppmann[106, p. 91] and is used without any changes. Even though the view is used within the view to edit semantic models, it belongs to the *Model Management Environment* component, because it does not allow to change models. A screenshot of this diagram can be seen in Figure 5.7 in Section 5.2.1.3.

5.2.1.3 Model Definition

In order to define the semantic models, the *Model Definition Environment* is used. This component also provides several AngularJS controllers and views, as well as modal dialogs. The different views and their functionality are discussed in this section along with some selected screenshots.

SemanticModelEditCtrl

The *SemanticModelEditCtrl* is one of the main components of the whole modeling environment. It enables the definition and refinement of semantic models. Hence, it has a dependency to the *SemanticModelContractService* and registers callback functions for different events. The *SemanticModelEditCtrl* can load and save semantic models to the back-end. For the process of defining semantic models, the remaining controllers and views of this section are incorporated. A screenshot, revealing the view of this controller is shown in Figure 5.6.

The view consists of three areas. On the left hand side, a visual representation of the semantic model is shown. Upon clicking on a specific type, the middle area depicts the type along with the corresponding attributes. In case an instance to that model already exists, the attributes are filled with the respective values of the first instance. Different instances can be selected from the object diagram in Figure 5.7. The right hand side provides control buttons, to add new types and to save or reset the model.

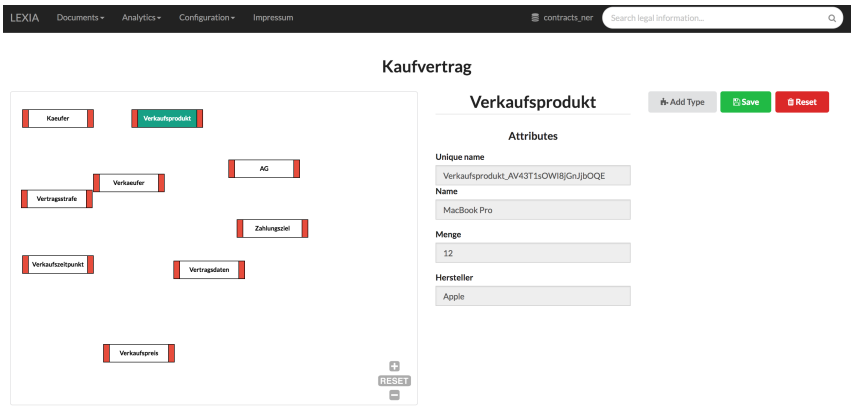


Figure 5.6: Screenshot of the model edit view

Source: Own screenshot

Figure 5.7: Screenshot of the object diagram

Source: Own screenshot

ModelCreateContractTypeCtrl

ModelCreateContractTypeCtrl represents a modal dialog which enables the user to add new types for a given semantic model. This modal view is invoked by clicking on the *Add Type* button in Figure 5.6. A screenshot of this modal view is depicted in Figure 5.8.

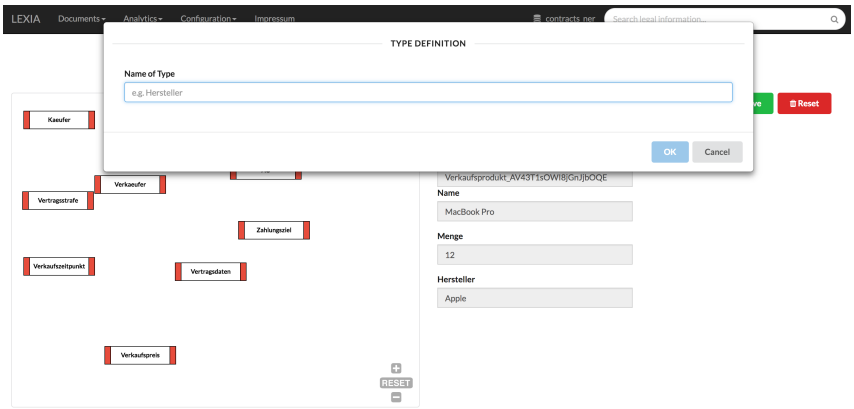


Figure 5.8: Modal view to create a model type

Source: Own screenshot

ModelCreateContractAttributesCtrl

This view is also a modal view that opens a dialog in which the user can create and modify attributes for a given type. Figure 5.9 shows a screenshot of this view. The dialog can be reached by selecting *Edit Attributes*, while right clicking on a type.

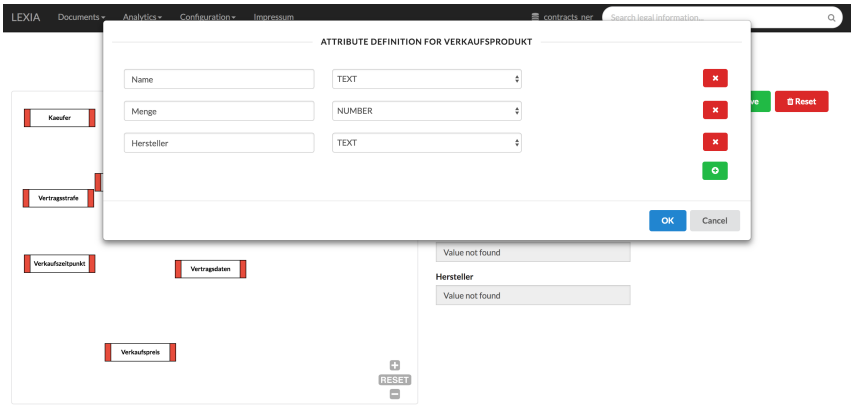


Figure 5.9: Modal view to create a type attribute

Source: Own screenshot

ModelContractSummaryCtrl

Upon double clicking on a type, the *ModelContractSummaryCtrl* view is opened modally. It shows a summary of all attributes of the given type, as shown in Figure 5.10.

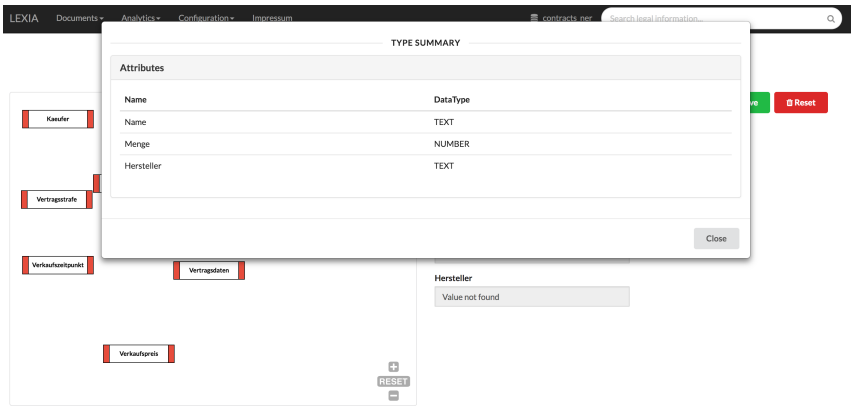


Figure 5.10: Modal view to summarize a given model type

Source: Own screenshot

5.2.2 Pipeline Execution

The final section of the chapter about implementing the component to semantically analyze and structure legal contracts deals with the UI to execute pipelines. In fact this component only consists of the *PipelineSelectionController*. However, the entry point for that process is the *ContractsController*. This controller was already implemented in Lexia in order to gain access to all contracts stored in the system. A screenshot of this view is provided in Figure 5.11.

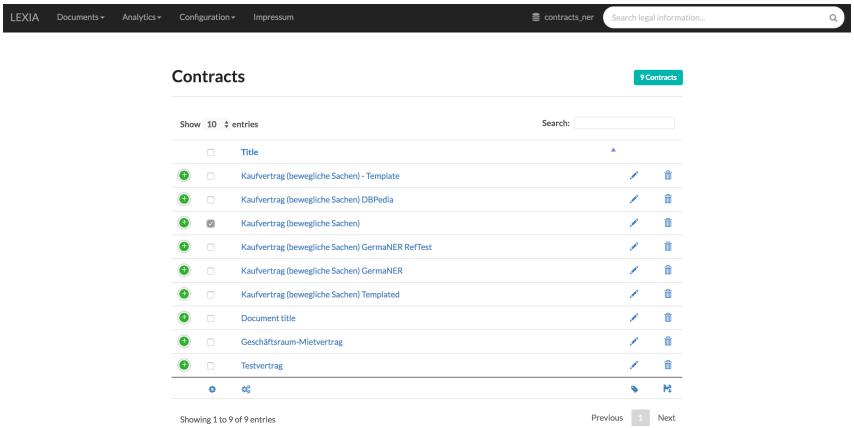


Figure 5.11: View to select a contract

Source: Own screenshot

Once the user selects at least one contract (several contracts can be selected as well), the *tag* icon is shown in the footer of the list. A click on this icon invokes the *PipelineSelectionController* view, shown in Figure 5.12.

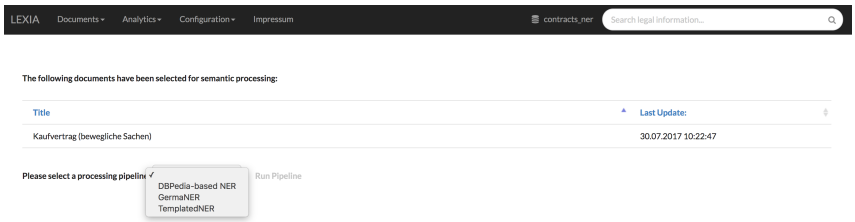


Figure 5.12: Screenshot of the pipeline selection view

Source: Own screenshot

This view reveals the selected document(s), which are subject to be processed by a NER pipeline. Furthermore, the user can select the preferred technique, depending on the pipelines available by the system. One selected, the *Run Pipeline* button shows up and hence, the pipeline can be executed. Figure 5.13 shows the view after a NER pipeline is finished successfully.

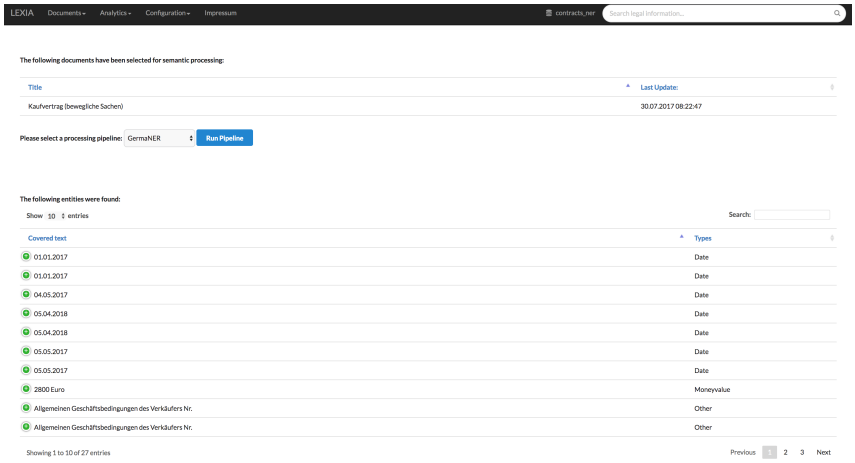


Figure 5.13: Screenshot of the pipeline selection view after execution

Source: Own screenshot

Depending on the selected NER approach, the linking options are shown as well after the pipeline is done. As shown in Figure 5.14,

The user must select one of the already processed contracts, which shall be used to disambiguate its NEs to a model. In terms of model selection, the user can either invoke an automatic model selection, or manually select the desired

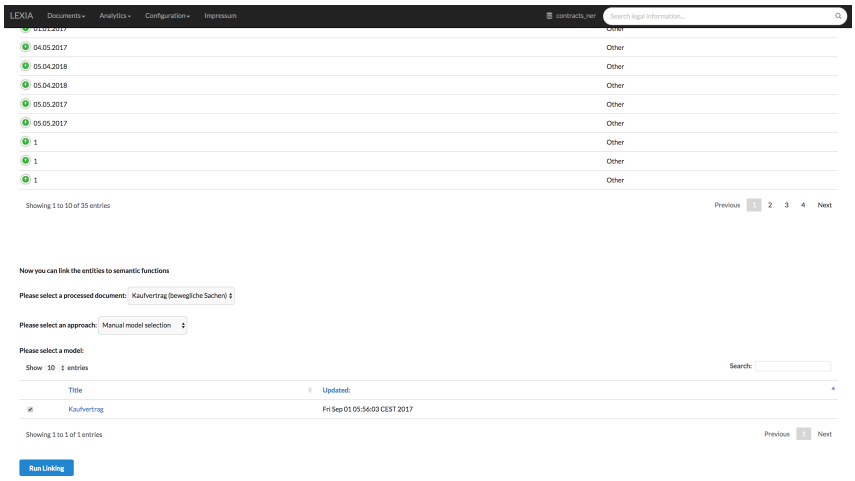


Figure 5.14: Screenshot of the view to configure the disambiguation

Source: Own screenshot

one. Afterwards the linking can be started. This set of screenshots represents the process incorporating templated NER and thus, only templated NED is available for selection. The results of the NED process, are shown in Figure 5.15.

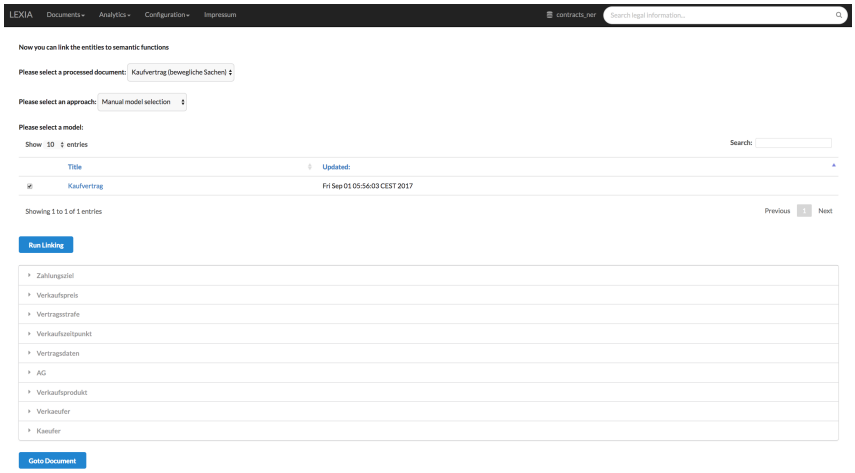


Figure 5.15: Screenshot of the view revealing the results of NED

Source: Own screenshot

In this final screen of the pipeline execution, the user can navigate directly to the processed document. Therefore the button *Goto Document* exists.

In Figure 5.16 the legal contract view is shown. The right area of this screen offers an accordion called *Semantic Model*. The structured contract is shown upon clicking on this button.

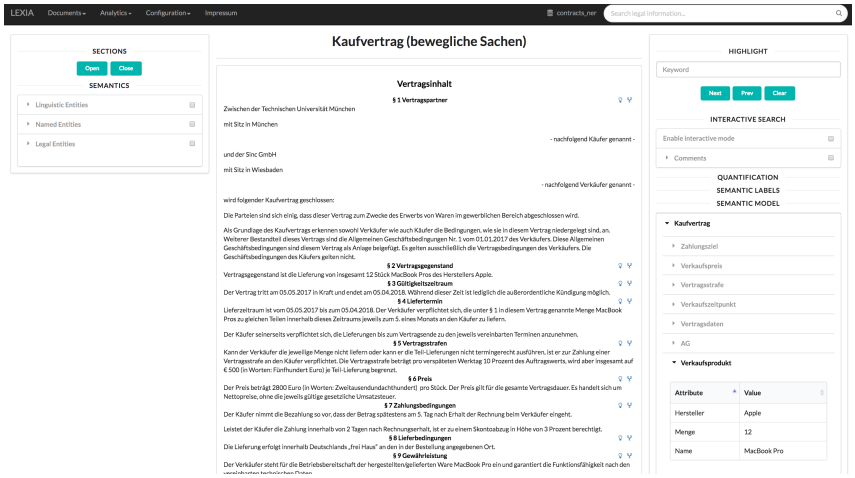


Figure 5.16: Screenshot of the contract view along with the structured semantic information

Source: Own screenshot

6 Evaluation

This evaluation chapter compromises two key sections. The first deals with a qualitative assessment of the prototypical implementation created in the course of this thesis. The second performs a quantitative evaluation of the different NER approaches, as well as of the NED technique incorporated in this implementation.

6.1 Qualitative

6.1.1 Idea

For the qualitative assessment of the prototypical implementation semi structured expert interviews are conducted. The main idea behind these interviews is to get detailed information about the usability and utility of the components being implemented. Furthermore, these interviews are used to obtain typical information demand of legal practitioners. Also typical problems from the view of legal experts can be experienced. This knowledge can help for future work, to tailor an implementation even better towards the user and business needs.

6.1.2 Interview Guideline

The interview guideline is revealed in Appendix C.1. It is split into four parts.

At first, the background of the interview partner is elicited. Who is the employer, which position does the partner hold, in which legal domain is the interview partner busy, and if there is some affinity to IT.

In the second part, questions to obtain the data on which the expert is working on are asked. This is necessary not just to evaluate whether the interview partner's answers are relevant for the evaluation, but also to learn about the information demand of such a legal expert. This knowledge can be utilized for future work a lot.

The third part specifically relates to IE and NER. The interview partner is asked whether he already utilizes tools to manage contracts and is already confronted with the results of this work. It is also elicited, whether a structured view of semantic information of a legal contract helps.

The final part of the interview consists of a live demo. This was used to gain valuable feedback about the actual implementation. Of course, this part was only conducted for person to person interviews, but not for phone interviews.

6.1.3 Selection of Interview Partners

The experts were selected mainly based on their occupational task. As long as the person is a legal practitioner, working a lot with contracts, the person was already a valuable resource. Different legal experts being project partners from previous projects were contacted, as well as random lawyers in the Munich area. At the end, three interviews were conducted.

6.1.4 Outcome

The experts mentioned that they are working on contracts about 70 to 80% of their working hours. This is pretty important for this study, in order to be able to gain valuable information. All three experts use already tools to support the management of legal contracts. However, these tools mainly regard to storage of documents, but not semantic analysis or other approaches, incorporating NLP.

In fact, all three experts had basically the same opinion about the developed system. They like the idea of recognizing NEs and using these to structure legal contracts. However, if only mundane structures can be created to reveal simple attributes like the name of the tenant or landlord in a rental agreement, it is not that useful. The reasoning behind this statement is, that a legal

expert is able to extract this information pretty fast by means of his trained eye. Extracting more detailed information such as the rental object along with properties like rental area or rent, would be already very helpful. Even more valuable would be the extraction and disambiguation of complex clauses such as liability clauses or change of control clauses.

Furthermore, the interview partners were in agreement about the difficulty to generalize the information demand of legal experts and lawyers. They told us that the relevant information heavily depends on the actual domain, but even more on the certain case. Even though all interview partners could exhibit decent IT affinity, they were not sure whether and how such an implementation could look like.

To summarize this qualitative evaluation, it can be said that such a system goes into proper direction, but further research is required to create more powerful and helpful system utilizing NER and NED. Such a development approach should be highly intertwined with incorporating legal experts to continuously exchange information and adapt the system based on the business needs.

6.2 Quantitative

After the qualitative evaluation already gave some nice insights, a quantitative assessment is required to properly determine the performance of the system, but also to be able to compare the different techniques against each other. Furthermore, such a quantitative evaluation is necessary for future work. Extensions on this study can easily be compared to the existing implementations based on the evaluation performed in this section.

6.2.1 NER

The first NLP task was to identify NEs. An introduction about the approaches to this topic is provided in Chapter 4.1. In the following, the evaluation method, the data used, as well as the results of the evaluation are described.

6.2.1.1 Evaluation Method

In terms of evaluating NER approaches, each of the three implemented techniques is first evaluated, before the obtained results are compared. Different evaluation metrics exist for the evaluation of NER systems[100]. The assessment is basically to check the system's ability on finding the boundaries of names and their correct types. When evaluating a NER system this way, there is one big problem that is caused by the fact that NER is a sequence labeling task[77].

Listing 6.1: Example to illustrate the problem with evaluating NER

```
The landlord of this flat is [PER Ingo Glaser].
```

The simple sentence from Listing 6.1 can easily demonstrate this issue. Only a single NE is contained in this sentence, that is *Ingo Glaser*. A system, recognizing only one of the two tokens *Ingo* as a NE has not fulfilled the task of NER perfectly. The question when evaluating such a system is, how to deal with such a partial NE recognition. Of course the problem can be also, that a system tags more tokens than it should, so the span exceeds the NE. Incorporating these boundary errors, can be quite challenging when evaluating NER systems. The evaluation in this work only approves a tagged span, when it is equal to the span enclosing the actual NE, with other words, perfect matching is required.

For the evaluation the state of the art approach of IR and IE has been used. This means that a confusion matrix is created for every approach. Based on this confusion matrix, each NE type is evaluated first[77]. For that the confusion matrix is used to count the *True Positives* (TP), the *False Positives* (FP), the *rue Negatives* (TN), and the *False Negatives* (FN) of each NE type. Hereby, the number of properly detected NEs is referred to TP, while annotations which do not reflect an actual NE, are said FP. FNs refer to NEs not being detected, and the remaining cases are the TN, that is all tokens not representing a NE and thus being not recognized as one. Those metrics are determined by creating a table of confusion for each category. Based on that, the precision, recall, and F_1 measure is calculated[104]. Afterwards the overall measures for each method are determined. Hereby the accuracy is not used, because it is quite superfluous for a NER system. Just a minority of all tokens

from a given text represent NEs. Thus, TN of such a system are very high relatively to the total amount of tokens[77, 117].

All categories used in this implementation are incorporated into the assessment: (1) person (PER), (2) organization (ORG), (3) location (LOC), (4) money value (MV), (5) date (DA), (6) reference (REF), and (7) other (OTH). For the evaluation, the following formulas are used.

$$precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$recall = \frac{TP}{TP + FN} \tag{6.2}$$

As shown in Equation 6.1, the precision is the ratio of the number of correctly labeled responses to the total labeled, while recall (Equation 6.2) is the ratio of the number of correctly labeled responses to the total that should have been labeled. Equation 6.3 depicts the definition of the F_1 measure, which is the harmonic mean of the two.

$$F_1 = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \tag{6.3}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{6.4}$$

Since $\beta^1 = 1$ is assumed for this evaluation, the formula in 6.4 represents the F_1 measure used in the remainder.

6.2.1.2 Data used

As already mentioned in Section 2.2, data sets for NER barely exist within the legal domain. As a consequence, the evaluation data set for this work has to be created manually. Since the focus of this work is semantically analyzing and structuring legal contracts, an evaluation corpus consisting of contracts would be a great fit. However, due to a lack of contracts in this study, the evaluation of the GermaNER pipeline and DBpedia Spotlight pipeline is performed on judgments. A corpus of 500 judgments from the law of tenancy of the 8th

Zivilsentat of the German BGH was downloaded from *Rechtsprechung im Internet*[38]. A random selection of 20 judgments out of this corpus constitute the evaluation dataset, used for this assessment. The data set consisted of 25.423 token. Since these judgments were not annotated, a gold standard was created by hand as well.

Table 6.1: Composition of evaluation data set

NE Types	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**O**
Count	114	106	45	267	78	310	182	24314

The composition of this data set is shown in Table 6.1. This distribution of NE types is pretty common for the legal domain. The abbreviations used in the table, are applied for the rest of this chapter, while *O* is referring to *not a NE*.

Templates do not exist for judgments and hence, the templated NER approach had to be evaluated on legal contracts. For this reason, 5 different contracts were selected: (1) a purchase agreement, (2)a lease contract, (3) an employment agreement, (4) a lease agreement for commercial premises, and (5) a GmbH contract. This ended up in a total of 7790 token, including the distribution of NEs as depicted in Table 6.2

Table 6.2: Composition of evaluation data set for templated NER

NE Types	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**O**
Count	14	8	23	38	23	25	46	7614

6.2.1.3 Assessment

In order to get a bit more accurate results, the evaluation was performed three rounds for each method. The average values for these three rounds was then used to answer the following two main questions:

1. Which implemented NER pipeline performs best?
2. Which NE type is recognized best?
3. Which NE type is recognized worst?

[38] `http://www.rechtsprechung-im-internet.de`

Before answering those questions, the results of each method are presented first.

GermaNER pipeline

The confusion matrix for the pipeline utilizing GermaNER is shown in Table 6.3.

Table 6.3: Confusion matrix for GermaNER implementation

		Gold							
	Tokens	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**O**
	PER	24	0	0	0	0	0	0	0
	ORG	0	59	2	0	0	0	0	0
	LOC	0	0	14	0	0	2	0	1
Response	**DA**	0	0	9	229	0	0	0	0
	MV	0	0	0	0	62	0	0	0
	REF	0	0	2	0	2	261	0	0
	OTH	6	9	9	2	7	1	44	7
	O	84	38	9	36	7	46	138	24306

The amount of FPs is over all categories, other than *other*, pretty small. This leads to the surprisingly high precisions, as shown in Table 6.4. The types *person*, *organization*, *date*, *money value*, and *reference* scored a very high precision around 1. This is not that surprising for the three last-mentioned types, because they are based on regex, which typically does not overreact. On the other side, a lot of NEs were not recognized at all, which explains the underwhelming recall values. Again, the rule-based approaches (*date*, *money value*, and *reference*) performed very well, but in particular the recognition of *persons* and *locations* was pretty bad, with a recall of 0.21, respectively 0.31. A nice observation is the fact, that the system did not classify too many NEs into the wrong type. Of course, the category *other* has the worst performance of all types. The reason behind this, may be that one functionality of that type is to intercept other types, which somehow fall through their own classifiers.

In summary it can be ascertained that the regex based recognitions worked very well with a F_1 measure between 0.89 and 0.91, while the types recognized by GermaNER itself only perform with a F_1 below 0.5, except for *organizations*.

Table 6.4: GermaNER implementation performance

		Precision	Recall	F_1
NE Type	**PER**	1	0.21	0.35
	ORG	0.97	0.56	0.71
	LOC	0.82	0.31	0.45
	DA	0.96	0.86	0.91
	MV	1	0.79	0.89
	REF	0.98	0.84	0.91
	OTH	0.52	0.24	0.33
	Overall	0.98	0.68	0.80

DBpedia Spotlight

Table 6.5 depicts the resulting confusion matrix, when evaluating the DBpedia Spotlight pipeline.

Table 6.5: Confusion matrix for DBpedia Spotlight implementation

		Gold							
	Tokens	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**O**
Response	**PER**	40	0	0	0	0	0	0	0
	ORG	0	67	2	0	0	0	0	0
	LOC	10	12	28	0	0	0	1	11
	DA	1	0	9	229	0	0	0	0
	MV	0	0	0	0	62	0	0	0
	REF	0	0	1	0	0	261	0	0
	OTH	32	25	3	2	8	3	165	137
	O	34	4	2	31	8	46	16	24193

The results of extracting rule-based *dates*, *money values*, and *references* are again pretty good, having a precision between 0.96 and 1. Some NEs of these types were missed, which is represented by a recall between 0.79 and 0.87. This is what one would expect when utilizing rule-based approaches for entities, being represented by rather simple structures. The three main classes performed quite differently. While *person* has a perfect precision (1), but a disappointing recall (0.34), *location* exhibits a poor precision (0.45) along with a semi recall (0.62). *Organization* reached the same recall, but a perfect precision of 0.97. The bad performance of recognizing *locations* is surprising, as one would expect that a knowledge base includes many locations. In fact, the result may be caused by the system's transformation of DBpedia types to the Lexia typesystem. The same reason could have caused the very low recall of

persons. The miscellaneous class has a decent recall (0.91), but a low precision of 0.44. A knowledge base such as DBpedia constitutes a huge variety of terms causing that recall, while the precision may be interfered by the transformation process, too. Nevertheless, the overall performance of this pipeline is typical for knowledge based systems, having a overall F_1 measure of 0.87.

Table 6.6: DBpedia Spotlight implementation performance

		Precision	**Recall**	$\boldsymbol{F_1}$
NE Type	**PER**	1	0.34	0.51
	ORG	0.97	0.62	0.76
	LOC	0.45	0.62	0.52
	DA	0.96	0.87	0.91
	MV	1	0.79	0.86
	REF	0.99	0.84	0.91
	OTH	0.44	0.91	0.59
	Overall	0.87	0.87	0.87

Templated

The resulting confusion matrix for the performance of templated NER is shown in Table 6.7.

Table 6.7: Confusion matrix for templatedNER

		Gold							
	Tokens	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**O**
Response	**PER**	11	0	0	0	0	0	0	0
	ORG	0	5	0	0	0	0	0	0
	LOC	0	3	16	0	0	0	1	0
	DA	0	0	0	31	0	0	5	2
	MV	0	0	0	0	18	0	1	1
	REF	0	0	0	0	0	19	0	0
	OTH	0	0	4	6	4	3	34	8
	O	3	0	3	1	1	3	5	7603

It is noticeable that the system only annotates a few token which do not represent any NE. Only two tokens are detected as a *date*, one as a *money value*, and eight as *other*, which actually do not constitute any NE. The reason behind this is that the contract templates designate almost each NE as a placeholder, because NEs are usually specific to a contract instance. However, sixteen NEs are not detected. Contracts include often terms expressing public

organizations ore other institutions which are fixed to all contract instances and thus are already concrete in the template. Those NEs cannot be identified by the templated NER system. Another conspicuous finding is the amount of different NE types being recognized as *other*. This is in the nature of how templated NER categorizes the NEs. The placeholder name used in the template is analyzed to determine the type of a NE. Obviously such a name does not always indicate the type and hence a lot of NEs are categorized wrong. Nevertheless, templated NER is performing very well. It has an overall F_1 measure of 0.92, which is pretty decent. The other F_1 measures for the different NE type performances are all comparable with state-of-the-art systems, as Table 6.8 reveals.

Table 6.8: Templated NER performance

		Precision	**Recall**	$\boldsymbol{F_1}$
NE Type	**PER**	1	0.79	0.88
	ORG	1	0.62	0.77
	LOC	0.80	0.84	0.82
	DA	0.82	0.91	0.86
	MV	0.9	0.86	0.88
	REF	1	0.86	0.93
	OTH	0.58	0.92	0.71
	Overall	0.94	0.91	0.92

The system even performed with a precision of 1 for the types *person*, *organization*, and *references*. But again, the good performances is based on the implementation of templated NER. A template is tailored to a specific set of contracts, which enables a very well NER, as long as only those NEs are of interest, which are used as placeholders. This evaluation basically regards two kinds of errors: (1) user errors and (2) system errors. The mistakes made by the system appear in the form of not recognizing a placeholder in the template or not being able to extract the context of a placeholder to identify the NE type. Implicitly also user errors are accounted, when the user does not properly name a placeholder and thus the system fails to determine the proper NE type.

This reasoning constraints the value of this evaluation quite a bit. It is not possible to compare a templated NER system with common NER tools. But the evaluation shows that the performance of this system is pretty good for its own purpose.

Comparison of the Results

(1) Which implemented NER pipeline performs best?

It is not common to compare three systems, whereas one of the systems was evaluated on a different evaluation data set. However, for this work it was not possible to evaluate all three approaches on the same data, as already mentioned in Section 6.2.1.2. Table 6.9 summarizes the results of this evaluation.

Table 6.9: NER performance of all three system over the evaluation data set

	Per-entity F_1							Overall		
System	**PER**	**ORG**	**LOC**	**DA**	**MV**	**REF**	**OTH**	**P**	**R**	F_1
Templated	.88	.77	.82	.86	.88	.93	.71	.94	.91	.92
GermaNER	.35	.71	.45	.91	.89	.91	.33	.98	.68	.8
DBpedia	.51	.76	.52	.91	.86	.91	.59	.87	.87	.87

The overall performance of templated NER (F_1) clearly exceeds the results of the pipelines incorporating GermaNER and DBpedia Spotlight. Comparing just the two latter, GermaNER reveals the better overall precision (0.98) over DBpedia Spotlight (0.87). This is not unexpectedly due to the fact that knowledge bases consist of a huge variety of different terms, which lead to the recognition of many tokens actually not representing any NE of interest. On the other side, the higher overall recall of DBpedia Spotlight (0.87 over 0.68), is not a surprise either, caused by the same fact. Hence, DBpedia Spotlight overall outperformed GermaNER (overall F_1 of 0.87 over 0.80). Nonetheless, templated NER is a very suitable and outstanding approach for NER on legal contracts, as long as templates exist. Both systems, GermaNER and DBpedia Spotlight were incorporated into a pipeline, but the system implemented in this work offers the possibility for errors as well. This leads to the assumption, that the two tools could improve way better, when evaluating independently from this system. When looking at different evaluations, such as the CoNNL-2013 shared task[11, 12] or the evaluation of DBpedia Spotlight [90], this assumption is partially confirmed.

Which NE type is recognized best?

The answer to this question can be easily given. The types being recognized via rule-based approaches (*date*, *money value*, and *reference*) obviously perform

the best. This is mainly caused due to the structure which represents types like those. Once those types are neglected, *organizations* perform the best.

Which NE type is recognized worst?

The type *other* has in its nature, that it not just compromises miscellaneous entities, it also often covers NE of other types, falling through their own classifiers. Moreover, there exist a huge variety of different NE types, excluding the set of categories used in this work. All those types shall be recognized by the *other* type. This may be feasible for a system such as DBpedia Spotlight, but statistical approaches and even the templated NER approach, clearly fail to detect all NEs of such types.

6.2.2 NED

In the course of this work, only templated NED is implemented. An evaluation based on measures such as precision, recall, and F_1 measure, as it has been done for NER approaches is not suitable at this point. The concept behind templated NED is quite simple, as discussed in Section 5.1.4. The disambiguation is solved by means of comparing the placeholder names in the template with the type and attribute names in the semantic model. This linking works always, as long as the user chooses the names accordingly. Hence, an error only occurs, if there is a mismatch between the naming in the semantic model and the contract template. It does not make sense to evaluate the person, who defined the evaluation set. One could suggest to conduct the evaluation by incorporating the whole process from the textual contract representation via NER and NED to the populated semantic model. However, only NEs recognized by the templated NER can be linked and thus, the evaluation result would mirror the results from assessing templatedNER. This is the reason, why no evaluation was performed on templated NED.

7 Discussion

Legal informatics is on a growing branch and one of the trends in recent IT developments. This is also accelerated by the digitalization. More and more legal documents are digitized and thus there is a big need of analyzing and structuring those. One important type of legal documents, which is used heavily by many different people, are contracts. Hence, this work was concerned with the semantic analyses and restructuring of legal contracts by utilizing NER and NED.

At first, a brief introduction was given to motivate the topic. Then, important terms used in this work were described. After a comprehensive literature review was performed, the relevant work for the field of NER and NED, also in particular for the legal domain in Germany, was summarized in the next chapter. The first part of this thesis was rounded off with the description of the research method used, as well as with the definition of relevant research questions.

The design chapter first discussed the various concepts of NER and NED. It can be distinguished between rule-based, knowledge-based, and ML-based approaches, augmented by a technique called templated NER, being developed in the course of this work. NED is mainly done by applying ML techniques, too. The literature differentiates between SML, SSML, and USML. Again, a templated approach was introduced as well. Furthermore, in the context of that chapter, the involved systems were introduced, namely Lexia and SocioCortex. Lexia is a legal data science environment that can be used for the exploration and analysis of legal documents. SocioCortex is a hybrid wiki with support for a powerful expression language. The design chapter was finished with an requirements analysis, eliciting crucial requirements for the prototypical implementation, before the definition of a suitable architecture took place. In the implementation phase of this thesis, Lexia was extended by a semantic analysis component, which allows the semantic analyses and restructuring of legal

contracts. For that reason, models which were transposed into SocioCortex are used to mirror the semantic information of such a contract.

The final part of this study constitutes the evaluation. A two-fold evaluation was applied. First, a qualitative assessment was conducted, by utilizing semi structured expert interviews. Second, a comprehensive quantitative evaluation was performed. Hereby, only NER was assessed, because there was no sense in evaluating the templated NED approach. A manually created data set was created to enable the evaluation. The evaluation was based on the state of the art approach for evaluating NER. A confusion matrix for each system was created to enable the determination of the relevant measures, namely precision, recall and F_1. By means of these metrics, the results could be obtained and compared.

What has just been described, was guided by individual research questions. These research questions were defined in Chapter 3.1. Through the course of this thesis, the research questions were investigated and answered. In what follows, the reflection on those questions is discussed in the next section. Next, a brief conclusion is made. This thesis is rounded up with an outlook into future research.

7.1 Reflection on the Research Questions

In the paragraphs that follow, a brief summary of the results of this work, referring to the individual research questions defined in Chapter 3.1, is provided.

Which information does a stakeholder want to extract from legal contracts?

Lawyers as well as legal experts usually work with contracts on a daily base. The inspection of such documents represents a decent part of their workloads. Simple meta informations like the landlord and tenant of a renting agreement are important, but also more complex knowledge such as the rental object along with its properties, including size, location or rent. A jurist with his trained eyes is pretty fast in extracting this information, in particular if the

contract design is familiar and standardized. A system as the one developed in the context of this thesis, must extract this information extremely fast and display it in a very structural way, otherwise it would not be beneficial to the user. Speaking about contracts in general, complex issues such as the question of liability are crucial, as well as various individual parameters for specific contract types, like change of control clauses. Extracting these information and presenting it to a user in a smart way would be totally helpful. Recapitulating it can be noticed, that the information demand of legal experts cannot be generalized, but needs to be tailored to the specific use case and domain.

What are the functional and non-functional requirements of a software for the analysis of legal contracts?

The FR and NFR were elicited in Chapter 4.4. Important FR for a system semantically analyzing and structuring legal contracts are to incorporate different methods to NER and NED. The more techniques are utilized, such as statistical SML based NER or knowledge based NER, the better the system can perform. It is almost always beneficial to resort to knowledge bases either way. A non-functional crucial aspect is the pipeline architecture. Utilizing an architecture like Apache UIMA's pipeline system simplifies the integration of different approaches a lot. Besides common NFRs like maintainability, reusability or extensibility, a smart UI is also favorable. Table 7.1 provides an overview of the defined requirements and their degree of fulfillment. Hereby, the requirements can be either completely fulfilled (✔), partially fulfilled (✓), or unfulfilled(✗).

As all the checkmarks indicate already, the majority of requirements is regarded and implemented. Only three FR are not implemented, each of them concerning different approaches to NED (FR04, FR05, and FR06). As this was just not possible in the course of this work, the violation of these requirements is not an issue. The implementation of templated NED (FR07) is included as it was planned. All approaches to NER are implemented as required (FR01, FR02, FR03, as well as NFR05, and NFR06). Apache UIMA Pipelines are used for all NER pipelines (FR08), only the combination of the pipelines is just possible within the UI, but not technically (FR09). The requirements concerning the semantic models were all minded (FR10-FR13). From a non-

Table 7.1: Verification of the requirements

ID	Requirement	Priority	Fulfillment
FR01	Statistical ML based NER	4	✓
FR02	Knowledge-based NER	4	✓
FR03	Templated NER	5	✓
FR04	Supervised NED	2	✗
FR05	Unsupervised NED	1	✗
FR06	Dictionary-based NED	2	✗
FR07	Templated NED	3	✓
FR08	Use of Apache UIMA Pipelines for NER	5	✓
FR09	Combine NER and NED Pipelines	3	✓
FR10	Create Contract Models	5	✓
FR11	Update and Change Contract Models	3	✓
FR12	Delete Contract Models	2	✓
FR13	Use of Lexia's Drafted Legal Documents	5	✓
NFR01	Simple UI	5	✓
NFR02	Maintainability of Software Architecture	5	✓
NFR03	Extensibility of Software Architecture	5	✓
NFR04	Reusability of Software Architecture	3	✓
NFR05	Use GermaNER for Statistical ML NER	4	✓
NFR06	Use DBPedia for Knowledge-based NER	4	✓
NFR07	Reuse the Models and Structure from the Semantic Model Compoenent	2	✓
NFR08	Incorporate Elasticsearch for the Storage	5	✓
NFR09	Incorporate SocioCortex for the Storage	3	✓
NFR10	Utilize Lexia's existing Pipeline Architecture	5	✓

function point of view, all requirements were heeded, because these are highly relevant for the architecture design.

Which NLP technologies can be used, to extract the semantic meaning of a legal contract? How to combine these technologies into an Apache UIMA pipeline?

For the semantic analysis of legal contracts, different NLP technologies are applicable. As this work has shown, a good starting point is the utilization of NER and NED technologies. These two tasks can be executed consecutively,

where NED picks the results from NER, to assign NEs to semantic functions or roles. In terms of NER, different methods are suitable. In particular SML-based methodologies are proper for this task. Techniques as HMM, CRF or VSMs proved to be appropriate approaches. A NER system should always incorporate knowledge bases, for instance in form of gazetteers. Coming to NED, methods based on SML or SSML are most helpful. However, when trying to assign NEs to individual contract models, the task is very challenging and further research is required. A lot of state of the art tools already provide support for Apache UIMA pipelines. But even if a system does not support it out of the box, a wrapper can be easily created.

How does a prototypical implementation enabling the semantic analysis of legal contracts look like?

In the framework of this thesis, a prototypical implementation to semantically analyses and structure legal contracts was developed. The implementation was integrated into Lexia, also by means of SocioCortex. The NER task can be performed by applying three different approaches: (1) GermaNER, (2) DBpedia Spotlight, and (3) templated NER. In particular when having contract templates, templated NER is a really suitable method. When the extracted NEs are subject to be linked to semantic functions, semantic models of legal contracts are a proper way to define these roles. The linking process to the types and attributes of such a model can be executed by applying NED. After recognizing NEs via templated NER, obviosuly an approach called templated NED is a good fit to do so. After the process has finished, a tabular depiction of the semantic knowledge seems to be a convenient method to reveal semantic information.

How can such a system be integrated into the workflow of potential stakeholders?

Legal experts often work together and communicate a lot. The reason behind this is in the nature of their workflow. They need to lookup old cases on a regular base and thus their work is based on huge corpora of legal documents. This demands for a legal data-science environment such as Lexia. Such a platform offers many ways to work in a collaborative manner. Hence, a platform

like Lexia is highly qualified to serve as a base for an implementation enabling semantic analysis of legal contracts. This is the reason, why the prototypical implementation of this study is integrated into Lexia.

7.2 Conclusion

In this work, a prototypical implementation enabling the semantic analysis and structuring of legal contracts was designed and developed, utilizing Lexia. Common concepts and strategies found in the literature study form the basis for the developed requirements and solutions. Three different NER methods, namely GermaNER, DBpedia Spotlight, as well as an individually developed solution called templated NER are responsible for the extraction of NEs. The disambiguation of the recognized entities towards semantic functions, which are represented in semantic models, is done by NED. When having individual domain specific models, it is very hard to incorporate proper NED. This is mainly because of the lack of training data. In order to attain the breakthrough from a legal contract to a populated contract model, this work implemented the templated NED approach. By means of this approach, a contract model in SocioCortex was successfully populated with semantic information within the contract.

The pipeline architecture is based on Apache UIMA and thus can be easily extended. This enables the integration of existing analysis engines, used in Lexia, into the pipelines for NER and NED. Future work on the semantic anlaysis of legal contracts, can be easily integrated into the existing pipeline architecture.

The evaluation of the different approaches used in this study showed, that templated NER is an appropriate approach to recognizing NEs within legal contracts, which are based on templates. It also revealed the applicability of common NER tools like GermaNER or DBpedia within the legal domain, but also showed the necessity of future research in this field.

The prototypical implementation along with the outcomes of this work are an additional knowledge base and provide an appropriate starting point for future research in the fields of NER and NED on German legal contracts.

7.3 Limitations and Feature Work

Even though this work provides a good starting point for further work, some limitations described must be kept in mind.

Although each evaluation experiment was conducted three times in order to obtain a significant result, and even though the results looked still quite promising, the evaluation experiments require further replication to attain a statistically significant value. This is caused in particular due to the manual creation of the evaluation date set, which is furthermore very small. Moreover, the evaluation of templated NER was obviously conducted on a different data set than the other two approaches (GermaNER and DBpedia Spotlight) and hence, the comparison of the three methods it not suitable.

The results of the GermaNER as well as the DBpedia Spotlight pipeline may not reflect their actual performance. The NE types, regarded in this work are: person, organization, location, date, money value, reference, and other. Dates, money values and references were only detected using rule-based methodologies, but incorporated into both pipelines. This already sophisticates the results. In addition, these two technologies were not used in isolation, but utilized by the prototypical implementation of this thesis. Hence, system errors are conveyed to the two tools.

The templated NER approach is only suitable for corpora, where a small amount of templates define a massive number of contracts. But if that is the case, by diligently defining the template placeholders and incorporating templated NED, awesome results can be achieved. Due to this, the implementation of the templated approaches to NER and NED are a promising approach for the semantic analysis and structuring of legal contracts. For the future work, it could be an interesting approach to train the models of NER tools, such as GermaNER specifically for the German legal domain. If at the same time, big evaluation data sets arise, the NER task on German legal contracts could be improved considerably. Then the next step, would be to build classifiers for the disambiguation of those recognized NEs, towards individual semantic models. Eventually, this may lead to digitized and properly structured legal contracts.

Bibliography

[1] Gianmaria Ajani, Guido Boella, Leonardo Lesmo, Marco Martin, Alessandro Mazzei, Daniele P Radicioni, and Piercarlo Rossi. Legal taxonomy syllabus version 2.0. *IDT*, page 9, 2009.

[2] Harith Alani, Sanghee Kim, David E Millard, Mark J Weal, Wendy Hall, Paul H Lewis, and Nigel R Shadbolt. Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18(1):14–21, 2003.

[3] Enrique Alfonseca and Suresh Manandhar. An unsupervised method for general named entity recognition and automated concept discovery. In *Proceedings of the 1st international conference on general WordNet, Mysore, India*, pages 34–43, 2002.

[4] Alias-i. Lingpipe 4.0.0, 2008.

[5] James F Allen. Natural language processing. 2003.

[6] Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 8–15. Association for Computational Linguistics, 2003.

[7] Tara Athan, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner. Legalruleml: Design principles and foundations. In *Reasoning Web International Summer School*, pages 151–188. Springer, 2015.

[8] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007.

[9] Laurie Bauer. *Introducing linguistic morphology*. Georgetown University Press, Washington, D.C., 2nd edition, 2003.

[10] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum entropy models for named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 148–151. Association for Computational Linguistics, 2003.

[11] Darina Benikova, Chris Biemann, Max Kisselew, and Sebastian Padó. Germeval 2014 named entity recognition shared task: companion paper. *Organization*, 7:281, 2014.

[12] Darina Benikova, Seid Muhie, Yimam Prabhakaran, and Santhanam Chris Biemann. C.: Germaner: Free open german named entity recognition tool. In *In: Proc. GSCL-2015*. Citeseer, 2015.

[13] Steven Bethard, Philip V Ogren, and Lee Becker. Cleartk 2.0: Design patterns for machine learning in uima. In *LREC*, pages 3289–3293, 2014.

[14] Chris Biemann, Claudio Giuliano, and Alfio Gliozzo. Unsupervised part-of-speech tagging supporting supervised methods. In *Proceedings of RANLP*, volume 7, pages 8–15, 2007.

[15] Chris Biemann and Martin Riedl. Text: Now in 2d! a framework for lexical expansion with contextual similarity. *Journal of Language Modelling*, 1(1):55–95, 2013.

[16] Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201. Association for Computational Linguistics, 1997.

[17] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227, 2009.

[18] Robert Blumberg and Shaku Atre. The problem with unstructured data. *Dm Review*, 13(42-49):62, 2003.

[19] Paul Bocij, Andrew Greasley, and Simon Hickie. *Business information systems: Technology, development and management*. Pearson education, 2008.

[20] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[21] Geert Booij. *The grammar of words: An introduction to linguistic morphology*. Oxford University Press, 2012.

[22] Andrew Borthwick and Ralph Grishman. *A maximum entropy approach to named entity recognition*. PhD thesis, New York University, Graduate School of Arts and Science, 1999.

[23] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Nyu: Description of the mene named entity system as used in muc-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7*. Citeseer, 1998.

[24] Joost Breuker and Rinke Hoekstra. Epistemology and ontology in core ontologies: Folaw and lri-core, two core ontologies for law. In *Proceedings of EKAW Workshop on Core ontologies. CEUR*, pages 19–24, 2004.

[25] Eric Brill. Part-of-speech tagging. In *Handbook of natural language processing*. CRC Press, 2000.

[26] Sergey Brin. Extracting patterns and relations from the world wide web. In *International Workshop on The World Wide Web and Databases*, pages 172–183. Springer, 1998.

[27] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[28] Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. Word-sense disambiguation using statistical methods. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 264–270. Association for Computational Linguistics, 1991.

[29] Razvan C Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *Eacl*, volume 6, pages 9–16, 2006.

[30] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57, 2014.

[31] Cristian Cardellino, Milagro Teruel, Laura Alonso Alemany, and Serena Villata. A low-cost, high-coverage legal named entity recognizer, classifier and linker. In *16th International Conference on Artificial Intelligence and Law (ICAIL-2017)*, 2017.

[32] Hai Leong Chieu and Hwee Tou Ng. Named entity recognition: a maximum entropy approach using global information. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.

[33] Laura Chiticariu, Yunyao Li, and Frederick R Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*, pages 827–832, 2013.

[34] Alexander Clark. Combining distributional and morphological information for part of speech induction. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 59–66. Association for Computational Linguistics, 2003.

[35] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, pages 100–110, 1999.

[36] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[37] Thomas M. Cover and A. Thomas Joy. Elements of information theory. *Wiley*, 4:10, 1991.

[38] Alessandro Cucchiarelli and Paola Velardi. Unsupervised named entity recognition using syntactic and semantic contextual evidence. *Computational Linguistics*, 27(1):123–131, 2001.

[39] Silviu Cucerzan. Large-scale named entity disambiguation based on wikipedia data. 2007.

[40] James R Curran and Stephen Clark. Language independent ner using a maximum entropy tagger. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 164–167. Association for Computational Linguistics, 2003.

[41] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 121–124. ACM, 2013.

[42] Emile de Maat and Radboud Winkels. Automated classification of norms in sources of law. In *Semantic Processing of Legal Texts*, pages 170–191. Springer, 2010.

[43] Hongbo Deng, Irwin King, and Michael R Lyu. Entropy-biased models for query representation on the click graph. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 339–346. ACM, 2009.

[44] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. Analysis of named entity recognition and linking for tweets. *Information Processing & Management*, 51(2):32–49, 2015.

[45] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997.

[46] Doug Downey, Matthew Broadhead, and Oren Etzioni. Locating complex named entities in web text. In *IJCAI*, volume 7, pages 2733–2739, 2007.

[47] Christopher Dozier, Ravikumar Kondadadi, Marc Light, Arun Vachher, Sriharsha Veeramachaneni, and Ramdev Wudali. Named entity recognition and resolution in legal text. In *Semantic Processing of Legal Texts*, pages 27–43. Springer, 2010.

[48] Richard O Duda and Peter E Hart. *Pattern Elesslfication and Scene Analysis*. Wiley, 1973.

[49] Micha Elsner, Eugene Charniak, and Mark Johnson. Structured generative models for unsupervised named-entity clustering. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the*

North American Chapter of the Association for Computational Linguistics, pages 164–172. Association for Computational Linguistics, 2009.

[50] Richard Evans and Stafford Street. A framework for named entity recognition in the open domain. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 260(267-274):110, 2003.

[51] Manaal Faruqui, Sebastian Padó, and Maschinelle Sprachverarbeitung. Training and evaluating a german named entity recognizer with semantic generalization. In *KONVENS*, pages 129–133, 2010.

[52] David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[53] David Ferrucci, Adam Lally, Karin Verspoor, and Eric Nyberg. Unstructured information management architecture (uima) version 1.0. *OASIS Standard, OASIS*, 2009.

[54] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[55] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 168–171. Association for Computational Linguistics, 2003.

[56] Luciano Floridi. Is semantic information meaningful data? *Philosophy and Phenomenological Research*, 70(2):351–370, 2005.

[57] William Gale, Kenneth Ward Church, and David Yarowsky. Estimating upper and lower bounds on the performance of word-sense disambiguation programs. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 249–256. Association for Computational Linguistics, 1992.

[58] William A Gale, Kenneth W Church, and David Yarowsky. A method for disambiguating word senses in a large corpus. *Computers and the Humanities*, 26(5):415–439, 1992.

[59] Aldo Gangemi, Maria-Teresa Sagri, and Daniela Tiscornia. A constructive framework for legal ontologies. In *Law and the semantic web*, pages 97–124. Springer, 2005.

[60] Matthias Grabmair, Kevin D Ashley, Ran Chen, Preethi Sureshkumar, Chen Wang, Eric Nyberg, and Vern R Walker. Introducing luima: an experiment in legal conceptual retrieval of vaccine injury decisions using a uima type system and tools. In *Proceedings of the 15th International Conference on Artificial Intelligence and Law*, pages 69–78. ACM, 2015.

[61] Ralph Grishman. The nyu system for muc-6 or where's the syntax? In *Proceedings of the 6th conference on Message understanding*, pages 167–175. Association for Computational Linguistics, 1995.

[62] Nizar Habash, Owen Rambow, and Ryan Roth. Mada+ tokan: A toolkit for arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. In *Proceedings of the 2nd international conference on Arabic language resources and tools (MEDAR), Cairo, Egypt*, volume 41, page 62, 2009.

[63] James Hammerton. Named entity recognition with long short-term memory. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 172–175. Association for Computational Linguistics, 2003.

[64] Xianpei Han and Jun Zhao. Structural semantic relatedness: a knowledge-based method to named entity disambiguation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 50–59. Association for Computational Linguistics, 2010.

[65] Elizabeth Hardcastle. *Business Information Systems*. Bookboon, 2008.

[66] Mustafa Hashmi. A methodology for extracting legal norms from regulatory documents. In *Enterprise Distributed Object Computing Workshop (EDOCW), 2015 IEEE 19th International*, pages 41–50. IEEE, 2015.

[67] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[68] Marti A Hearst. Uis for faceted navigation: Recent advances and remaining open problems. In *Proc. 2008 Workshop on Human-Computer Interaction and Information Retrieval*, 2008.

[69] Umesh R Hodeghatta and Umesh Nayak. Unsupervised machine learning. In *Business Analytics Using R-A Practical Approach*, pages 161–186. Springer, 2017.

[70] Guilherme Hoefel and Charles Elkan. Learning a two-stage svm/crf sequence classifier. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 271–278. ACM, 2008.

[71] Rinke Hoekstra, Joost Breuker, Marcello Di Bello, and Alexander Boer. The lkif core ontology of basic legal concepts. *LOAIT*, 321:43–63, 2007.

[72] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[73] Eduard Hovy, Roberto Navigli, and Simone Paolo Ponzetto. Collaboratively built semi-structured content and artificial intelligence: The story so far. *Artificial Intelligence*, 194:2–27, 2013.

[74] Christian Hänig, Stefan Bordag, and Stefan Thomas. Modular classifier ensemble architecture for named entity recognition on low resource systems. In *Workshop Proceedings of the 12th Edition of the KONVENS Conference*, pages 113–116, 2014.

[75] Peter Jackson and Isabelle Moulinier. *Natural language processing for online applications: Text retrieval, extraction and categorization*, volume 5. John Benjamins Publishing, 2007.

[76] Heng Ji and Ralph Grishman. Data selection in semi-supervised learning for name tagging. In *Proceedings of the Workshop on Information Extraction Beyond The Document*, pages 48–55. Association for Computational Linguistics, 2006.

[77] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[78] Dan Klein, Joseph Smarr, Huy Nguyen, and Christopher D Manning. Named entity recognition with character-level models. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 180–183. Association for Computational Linguistics, 2003.

[79] George R Krupka and Kevin Hausman. Isoquest inc.: Description of the netowl (tm) extractor system as used for muc-7. In *Proceedings of MUC*, volume 7, 1998.

[80] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[81] Kenneth C Laudon and Jane Price Laudon. *Essentials of management information systems*. Pearson Upper Saddle River, 2011.

[82] Elizabeth D Liddy. Natural language processing. 2001.

[83] Brian William Locke. Named entity recognition: Adapting to microblogging. 2009.

[84] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*, volume 999. MIT Press, 1999.

[85] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O'Reilly Media, Inc.", 2011.

[86] Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid wikis: Empowering users to collaboratively structure information. *ICSOFT (1)*, 11:250–259, 2011.

[87] James Mayfield, Paul McNamee, and Christine Piatko. Named entity recognition using hundreds of thousands of features. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 184–187. Association for Computational Linguistics, 2003.

[88] Andrew McAfee, Erik Brynjolfsson, and Thomas H Davenport. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[89] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[90] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM, 2011.

[91] Pablo N Mendes, Alexandre Passant, Pavan Kapanipathi, and Amit P Sheth. Linked open social signals. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 224–231. IEEE, 2010.

[92] Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition without gazetteers. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 1–8. Association for Computational Linguistics, 1999.

[93] Igor Minakov, George Rzevski, Petr Skobelev, and Simon Volman. Creating contract templates for car insurance using multi-agent based text understanding and clustering. In *Holonic and Multi-Agent Systems for Manufacturing*, pages 361–370. Springer, 2007.

[94] Tom Mitchell. Machine learning. macgraw-hill companies. *Inc*, 1:997, 1997.

[95] Andrea Moro and Roberto Navigli. Integrating syntactic and semantic analysis into the open information extraction paradigm. In *IJCAI*, pages 2148–2154, 2013.

[96] Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014.

[97] Johannes Muhr. *Design, Prototypical Implementation, and Evaluation of an Active Machine Learning Service in the Context of Legal Text Classification*. Master thesis, TU München, 2017.

[98] Eugene W Myers. An o (nd) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.

[99] Stefan Müller and Stephen Wechsler. Lexical approaches to argument structure. *Theoretical Linguistics*, 40(1-2):1–76, 2014.

[100] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[101] David Nadeau, Peter Turney, and Stan Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. *Advances in artificial intelligence*, pages 266–277, 2006.

[102] Roberto Navigli and Simone Paolo Ponzetto. Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250, 2012.

[103] Hien Nguyen and Tru Cao. Named entity disambiguation: A hybrid statistical and rule-based incremental approach. *The Semantic Web*, pages 420–433, 2008.

[104] Damien Nouvel, Maud Ehrmann, and Sophie Rosset. Evaluating named entity recognition. *Named Entities for Computational Linguistics*, pages 111–129.

[105] Naoaki Okazaki. Crfsuite: a fast implementation of conditional random fields (crfs). 2007.

[106] Dominik Oppmann. *Possibilities and Limitations of the Structured Transposition of Normative Texts in Functions on Typed Data Structures*. Master thesis, TU München, 2016.

[107] Marius Pasca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits, and Alpa Jain. Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge. In *AAAI*, volume 6, pages 1400–1405, 2006.

[108] Georgios Petasis, Frantz Vichot, Francis Wolinski, Georgios Paliouras, Vangelis Karkaletsis, and Constantine D Spyropoulos. Using machine learning to maintain rule-based named-entity recognition and classification systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 426–433. Association for Computational Linguistics, 2001.

[109] Pir Abdul Rasool Qureshi, Nasrullah Memon, and Uffe Kock Wiil. Languagenet: A novel framework for processing unstructured text information. In *Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on*, pages 95–100. IEEE, 2011.

[110] Delip Rao, Paul McNamee, and Mark Dredze. Entity linking: Finding extracted entities in a knowledge base. In *Multi-source, multilingual information extraction and summarization*, pages 93–115. Springer, 2013.

[111] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[112] Nils Reimers, Judith Eckle-Kohler, Carsten Schnober, Jungi Kim, and Iryna Gurevych. Germeval-2014: Nested named entity recognition with neural networks. In *Workshop Proceedings of the 12th Edition of the KONVENS Conference*, pages 117–120, 2014.

[113] Thomas Reschenhofer, Manoj Bhat, Adrian Hernandez-Mendez, and Florian Matthes. Lessons learned in aligning data and model evolution in collaborative information systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 132–141. ACM, 2016.

[114] Matthias Richter, Uwe Quasthoff, Erla Hallsteinsdóttir, and Chris Biemann. Exploiting the leipzig corpora collection. *Proceedings of the IS-LTC*, pages 68–73, 2006.

[115] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[116] Denise M Rousseau and Martin M Greller. Human resource practices: Administrative contract makers. *Human Resource Management*, 33(3):385–401, 1994.

[117] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: language-independent named entity recognition, 2003.

[118] M Saravanan, Balaraman Ravindran, and S Raman. Improving legal information retrieval using an ontological framework. *Artificial Intelligence and Law*, 17(2):101–124, 2009.

[119] Hinrich Schütze. Automatic word sense discrimination. *Computational linguistics*, 24(1):97–123, 1998.

[120] Satoshi Sekine. Nyu: Description of the japanese ne system used for met-2. In *Proc. Message Understanding Conference*, 1998.

[121] Satoshi Sekine and Chikashi Nobata. Definition, dictionaries and tagger for extended named entity hierarchy. In *LREC*, pages 1977–1980. Lisbon, Portugal, 2004.

[122] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[123] Claude E Shannon. Prediction and entropy of printed english. *Bell Labs Technical Journal*, 30(1):50–64, 1951.

[124] Yusuke Shinyama and Satoshi Sekine. Named entity discovery using comparable news articles. In *Proceedings of the 20th international conference on Computational Linguistics*, page 848. Association for Computational Linguistics, 2004.

[125] Jasmeet Singh and Vishal Gupta. Text stemming: Approaches, applications, and challenges. *ACM Computing Surveys (CSUR)*, 49(3):45, 2016.

[126] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[127] Mihai Surdeanu, Ramesh Nallapati, and Christopher Manning. Legal claim identification: Information extraction with hierarchically labeled data. In *Workshop Programme*, page 22, 2010.

[128] A Svyatkovskiy, K Imai, M Kroeger, and Y Shiraito. Large-scale text processing pipeline with apache spark. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 3928–3935. IEEE, 2016.

[129] Clemens Szyperski, Jan Bosch, and Wolfgang Weck. Component-oriented programming. *Lecture Notes in Computer Science*, 1743:184–184, 1999.

[130] A. Valente. Legal knowledge engineering: A modelling approach, frontiers in artificial intelligence and applications, 1996.

[131] Christian Veith, Michael Bandlow, Michael Harnisch, Hariolf Wenzler, Markus Hartung, and Dirk Hartung. How legal technology will change the business of law, 2016.

[132] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49(120):11, 1995.

[133] R Hevner Von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[134] Hanna M Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984. ACM, 2006.

[135] Stephan Walter. Definition extraction from court decisions using computational linguistic technology. *Formal Linguistics and Law*, 212:183, 2009.

[136] B Waltl, J Landthaler, E Scepankova, F Matthes, T Geiger, C Stocker, and C Schneider. Automated extraction of semantic information from german legal documents. In *IRIS: Internationales Rechtsinformatik Symposium*, Salzburg, Austria, 2017. IRIS.

[137] Bernhard Waltl, Florian Matthes, Tobias Waltl, and Thomas Grass. Lexia: A data science environment for semantic analysis of german legal texts. *Jusletter IT*, 2016.

[138] Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS quarterly*, pages xiii–xxiii, 2002.

[139] Geoffrey Weglarz. Two worlds of data unstructured and structured. *Information Management*, 14(9):19, 2004.

[140] Adam Wyner, Raquel Mochales-Palau, Marie-Francine Moens, and David Milward. Approaches to text mining arguments from legal cases. In *Semantic processing of legal texts*, pages 60–79. Springer, 2010.

[141] David Yarowsky. Word-sense disambiguation using statistical models of roget's categories trained on large corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 454–460. Association for Computational Linguistics, 1992.

Appendix

A Existing Lexia Classes

This appendix contains the full source code of existing Lexia classes, which are heavily used in this work.

A.1 PipelineRepository

```
public class PipelineRepository {

    private static PipelineRepository pipelineRepository = new PipelineRepository();

    private List<PipelineDescription> pipelines = Arrays.asList(
            pipe("Advanced Pipeline", AdvancedPipeline::new),
            pipe("Subject Pipeline", SubjectPipeline::new, false),
            pipe("Remove Annotations", RemoveAnnotationsPipeline::new),
            pipe("Reference Detection Pipeline", ReferenceDetectionPipeline::new),
            pipe("Active Learning Pipeline", ActiveLearningPipeline::new),
            pipe("Dependency Parsing Pipeline", DependencyParsingPipeline::new, false),
            pipe("Judgment Reference Pipeline", JudgmentReferencePipeline::new),
            pipe("Extract Syntactic Dependencies", SyntacticDependenciesPipeline::new,
                false),
            pipe("LSH Pipeline", LSHPipeline::new, false),
            pipe("GermaNER Pipeline", GermaNERPipeline::new, false),
            pipe("DBPedia Pipeline", DBPediaPipeline::new, false),
            pipe("Templated NER", TemplatedNERPipeline::new, false)

    );

    private PipelineRepository() {
        // inits the ids
        for (int i = 0; i < pipelines.size(); i++) {
            pipelines.get(i).id = i;
        }
    }

    public static PipelineRepository getInstance() {
        return pipelineRepository;
    }
```

```
31
32      public Pipeline getPipeline(int id) {
33          return pipelines.get(id).supplier.get();
34      }
35
36      public List<PipelineDescription> getPipelines() {
37          return pipelines;
38      }
39
40      public PipelineDescription getDesription(int id) {
41          return pipelines.get(id);
42      }
43
44      public String pipelineName(int id) {
45          return pipelines.get(id).name;
46      }
47
48      private PipelineDescription pipe(String name, Supplier<Pipeline> supplier) {
49          return new PipelineDescription(name, supplier);
50      }
51
52      private PipelineDescription pipe(String name, Supplier<Pipeline> supplier, boolean
            supportsParallel) {
53          return new PipelineDescription(name, supplier, supportsParallel);
54      }
55
56      public static class PipelineDescription {
57          private int id;
58          private String name;
59          private boolean supportsParallel;
60          private Supplier<Pipeline> supplier;
61
62          PipelineDescription(String name, Supplier<Pipeline> supplier) {
63              this.name = name;
64              this.supplier = supplier;
65              this.supportsParallel = true;
66          }
67
68          PipelineDescription(String name, Supplier<Pipeline> supplier, boolean
                supportsParallel) {
69              this(name, supplier);
70              this.supportsParallel = supportsParallel;
71          }
```

```
72
73          public boolean supportsParallel() {
74              return supportsParallel;
75          }
76
77          public int getId() {
78              return id;
79          }
80
81          public String getName() {
82              return name;
83          }
84      }
85  }
```

A.2 Abstract Pipeline

```
1  public abstract class Pipeline {
2      protected AnalysisEngine pipe;
3      protected JCas jCas;
4
5      @SafeVarargs
6      protected static AnalysisEngine assemblePipeline(String[] rutaScripts, Class<?
           extends AnalysisComponent>... c) throws ResourceInitializationException,
           IOException, InvalidXMLException {
7          AnalysisEngineDescription[] d = new AnalysisEngineDescription[c.length];
8
9          for (int i = 0; i < c.length; i++) {
10             d[i] = createEngineDescription(c[i]);
11         }
12
13         AnalysisEngineDescription componentsDesc = createEngineDescription(d);
14         AnalysisEngineDescription rutaDesc;
15         AnalysisEngineDescription completeDesc;
16         if (rutaScripts != null && rutaScripts.length > 0) {
17             rutaDesc =
                   createEngineDescription(UimaUtil.createRutaDescriptions(rutaScripts));
18             completeDesc = createEngineDescription(componentsDesc, rutaDesc);
19         } else
20             completeDesc = componentsDesc;
21         return createEngine(completeDesc);
22     }
23
24     public void setup(LegalDocument legalDocument, String[] rutaScripts) throws
           ResourceInitializationException, IOException, InvalidXMLException,
           CASException {
25         pipe = assemblePipeline(legalDocument, rutaScripts);
26         initCas(legalDocument, rutaScripts);
27     }
28
29     protected abstract AnalysisEngine assemblePipeline(LegalDocument legalDocument,
           String[] rutaScripts) throws ResourceInitializationException, IOException,
           InvalidXMLException;
```

```
31    protected void initCas(LegalDocument legalDocument, String[] rutaScripts) throws
          ResourceInitializationException, IOException, InvalidXMLException,
          CASException {
32        jCas = UimaUtil.produceJCas(rutaScripts);
33        jCas.setDocumentLanguage(legalDocument.getLanguage());
34    }
35
36    public PipelineResult process(Article article, String text) throws
          AnalysisEngineProcessException {
37        pipe.process(jCas);
38        return createAnnotationStructures(article);
39    }
40
41    public PipelineResult processStandalone(Article article, String text, String[]
          rutaScripts) throws AnalysisEngineProcessException, CASException,
          ResourceInitializationException, InvalidXMLException, IOException {
42        setup(article.getLegalDocument(), rutaScripts);
43        preArticle(article, text);
44        process(article, text);
45        postArticle(article);
46        return createAnnotationStructures(article);
47    }
48
49    protected PipelineResult createAnnotationStructures(Article article) {
50        Map<String, Object> result = new HashMap<>();
51        if (article != null) {
52            boolean isRawContentInHtml = HtmlUtil.isHtml(article.getContent());
53            result = createAnnotationAndAnnotationStructure(jCas,
                  isRawContentInHtml, article.getContent(), new JSONObject(),
                  article.getID(), article.getLegalDocument().getID());
54            storeAnnotationsCount(article.getLegalDocument(), jCas);
55        }
56        return new PipelineResult(result);
57    }
58
59    public void preDocument(LegalDocument d) {
60    }
61
62    public void postDocument(LegalDocument d) {
63        if (pipe != null)
64            pipe.destroy();
65    }
```

```
66
67      public void preArticle(Article article, String text) {
68          if (jCas != null) {
69              jCas.setDocumentLanguage(article.getLegalDocument().getLanguage());
70              jCas.setDocumentText(text);
71          }
72
73      }
74
75      public void postArticle(Article article) {
76          if (jCas != null)
77              jCas.reset();
78      }
79
80      public Predicate<Article> processArticle() {
81          return x -> true;
82      }
83  }
```

A.3 PipelineExecutor

```
1  public class PipelineExecutor {
2
3      private static PipelineJobList jobs = PipelineJobList.getInstance();
4      private static PipelineRepository repo = PipelineRepository.getInstance();
5      private int threads;
6      private BiFunction<List<Article>, Integer, List<List<Article>>> splitStrategy;
7      private int threadThreshold = 100;
8
9      public PipelineExecutor() {
10         // Split the article list into chunks the same size for each thread
11         this.splitStrategy = (x, y) -> {
12             int partitionSize = x.size() / y;
13             List<List<Article>> partitions = new LinkedList<>();
14             for (int i = 0; i < x.size(); i += partitionSize) {
15                 partitions.add(x.subList(i,
16                         Math.min(i + partitionSize, x.size())));
17             }
18             return partitions;
19         };
20         this.threads = 1;
21     }
22
23     public PipelineExecutor(int threads) {
24         this();
25         this.threads = threads;
26
27     }
28
29     public PipelineExecutor(int threads, BiFunction<List<Article>, Integer,
           List<List<Article>>> splitStrategy) {
30         this.threads = threads;
31         this.splitStrategy = splitStrategy;
32     }
33
34     public LegalDocument runPipeline(LegalDocument ld, String[] scripts, int pipeline,
           Predicate<Article> processArticle, boolean updateMetricInformation) throws
           AnalysisEngineProcessException, CASException,
           ResourceInitializationException, InvalidXMLException, IOException {
```

```
35          // init stuff
36          PipelineJob job = initJob(ld);
37          long startTime = System.currentTimeMillis();
38          JSONObject annotationStructures = new JSONObject();
39
40          List<Article> articlesToProcess = ld.getAllArticlesInArticleContainer(null);
41
42          if (articlesToProcess.size() > threadThreshold &&
                repo.getDesription(pipeline).supportsParallel())
43              threads = Runtime.getRuntime().availableProcessors(); // TODO Maybe
                    limit the threadcount?
44
45          Logger.info(String.format("Starting to annotate %s from %s with %s and %d
                thread(s)", ld.title, ld.promulgationDate, repo.pipelineName(pipeline),
                threads));
46
47          List<ExecutionUnit> executionUnits = new ArrayList<>(threads);
48          List<List<Article>> partitions = splitStrategy.apply(articlesToProcess,
                threads);
49
50          // Start threads
51          for (int i = 0; i < threads; i++) {
52              ExecutionUnit executionUnit = new ExecutionUnit(pipeline, scripts,
                    partitions.get(i), processArticle, job);
53              executionUnits.add(executionUnit);
54              executionUnit.start();
55          }
56
57          // Wait for completion of pipelines and merge annotationStructures
58          for (ExecutionUnit executionUnit : executionUnits) {
59              try {
60                  executionUnit.join();
61                  updateAnnotationStructures(annotationStructures,
                        executionUnit.annotationStructures);
62              } catch (InterruptedException e) {
63                  e.printStackTrace();
64              }
65          }
66
67          ld.annotationStructures = annotationStructures.toString();
68          if (updateMetricInformation)
69              ld.determineMetrics();
70          writeAnnotationData(ld);
```

```
71
72          long processingTime = System.currentTimeMillis() - startTime;
73          long startSave = System.currentTimeMillis();
74          articlesToProcess.forEach(Article::saveEntity);
75          ld.saveEntity();
76
77          long savingTime = System.currentTimeMillis() - startSave;
78
79
80          Logger.info("Finished! It took: " + (System.currentTimeMillis() - startTime)
                + "ms (Processing: " + processingTime + "ms, Saving: " + savingTime +
                "ms)");
81          job.close(); // Doesn't show count of processed articles anymore.
                Maybe-to-do?
82          return ld;
83      }
84
85      private boolean contentEmpty(Article article) {
86          return article.getContent() == null || article.getContent().isEmpty() ||
87                  HtmlUtil.convertToPlaintext(article.getContent()).isEmpty();
88      }
89
90      private void updateJob(PipelineJob job, Article article) {
91          job.setStatus("Processing \"Paragraph" + (article.getNr() == null ?
                article.weight : article.getNr()) + " " + article.getHeader() + "\"");
92          job.setCurrentArticle(job.getCurrentArticle() + 1);
93      }
94
95      private void updateAnnotationStructures(JSONObject annotationStructures,
            JSONObject newJson) {
96          Iterator e = newJson.keys();
97          while (e.hasNext()) {
98              String next = (String) e.next();
99              try {
100                 if (!annotationStructures.has(next))
101                     annotationStructures.put(next, newJson.get(next));
102             } catch (JSONException e1) {
103                 e1.printStackTrace();
104                 break;
105             }
106         }
107     }
108
109     private void writeAnnotationData(LegalDocument legalDocument) {
110         // create annotation dump
```

```
111         StringBuilder sb = new StringBuilder();
112         for (Article article : legalDocument.getAllArticlesInArticleContainer(null)) {
113             sb.append(createAnnotationDataForCSVDownload(article.weight,
                        article.getNr(), article.getAnnotations()));
114         }
115
116         legalDocument.annotationDataForDownload =
                "Number;Artikel;Annotation;Start;Ende;Text;;_;" + sb.toString();
117         if (legalDocument.SC_TYPE().equals(DraftedDocument.SC_TYPE)) {
118             writeAnnotationDataToFile(legalDocument, legalDocument.creationDate,
                        legalDocument.annotationDataForDownload.replace(";_;",
                        System.getProperty("line.separator")));
119         } else {
120             writeAnnotationDataToFile(legalDocument,
                        legalDocument.promulgationDate,
                        legalDocument.annotationDataForDownload.replace(";_;",
                        System.getProperty("line.separator")));
121         }
122     }
123
124     private PipelineJob initJob(LegalDocument legalDocument) {
125         PipelineJob job = new PipelineJob(legalDocument.getID(),
                legalDocument.getID(), legalDocument.getDocumentType(), "", new Date());
126
127         jobs.addJob(job);
128         job.setDocumentName(legalDocument.title);
129         int articleCount = legalDocument.getAllArticlesInArticleContainer(null).size();
130         job.setMaxArticles(articleCount);
131
132         return job;
133     }
134
135     private class ExecutionUnit extends Thread {
136         private final LegalDocument legalDocument;
137         private Pipeline pipeline;
138         private List<Article> articles;
139         private Predicate<Article> processArticle;
140         private PipelineJob job;
141         private PipelineResult pipelineResult;
142
143         private JSONObject annotationStructures;
144
145         public ExecutionUnit(int pipeline, String[] scripts, List<Article> articles,
                Predicate<Article> processArticle, PipelineJob job) throws CASException,
                ResourceInitializationException, InvalidXMLException, IOException {
146             this.articles = articles;
```

```
146             this.articles = articles;
147             this.processArticle = processArticle;
148             this.job = job;
149             this.legalDocument = articles.get(0).getLegalDocument();
150             this.annotationStructures = new JSONObject();
151 
152             this.pipeline = repo.getPipeline(pipeline);
153             this.pipeline.setup(legalDocument, scripts);
154         }
155 
156         @Override
157         public void run() {
158             pipeline.preDocument(legalDocument);
159             for (Article article : articles) {
160                 String plainText = HtmlUtil.isHtml(article.getContent()) ?
                        HtmlUtil.convertToPlaintext(article.getContent()) :
                        article.getContent();
161 
162                 if (!processArticle.test(article)
163                         || !pipeline.processArticle().test(article)
164                         || contentEmpty(article)
165                         || plainText.isEmpty())
166                     continue;
167 
168                 updateJob(job, article); // TODO jobs aren't threadsafe. Fix this.
169                 Logger.info("Processing Paragraph" + (article.getNr() == null ?
                        article.weight : article.getNr()) + " " + article.getHeader());
170                 pipeline.preArticle(article, plainText);
171                 try {
172                     pipelineResult = pipeline.process(article, plainText);
173                 } catch (AnalysisEngineProcessException e) {
174                     e.printStackTrace();
175                 }
176                 pipeline.postArticle(article);
177                 article.updateAnnotations(pipelineResult.getAnnotationData());
178 
179                 // merge annotation structures
180                 updateAnnotationStructures(annotationStructures,
                        pipelineResult.getAnnotationStructures());
181             }
182             pipeline.postDocument(legalDocument);
183         }
184     }
185 }
```

A.4 Entity

```
1  public abstract class Entity {
2
3      public abstract String getID();
4
5      public abstract void setID(String UID);
6
7      protected abstract boolean saveEntityElasticsearch();
8
9      @JsonIgnore
10     public abstract boolean isNewEntity();
11
12     public abstract String SC_TYPE();
13
14     public abstract String SC_TYPE_PLURAL();
15
16     public boolean saveEntity() {
17         return saveEntityElasticsearch();
18     }
19
20     public void deleteEntity() {
21         if (!isNewEntity()) {
22             ElasticsearchServer.deleteById(SC_TYPE(), getID());
23         }
24     }
25 }
```

B Newly implemented Classes

This appendix compromises different classes being developed in the course of this work, which were too big to put them in place.

B.1 CoNNLSegmenter

```
public class CoNLLSegmenter extends JCasAnnotator_ImplBase {

    public static final String FULL_VIEW = "FullView";

    @Override
    public void process(JCas jCas) throws AnalysisEngineProcessException {
        try {
            String text = jCas.getView(FULL_VIEW).getDocumentText();

            String plainText = text;

            plainText = plainText.replaceAll("[^A-Za-z0-9äöüÄÖÜß\n.]", "\n$0\n");

            plainText = plainText.replaceAll("\n{1,}|\r\n{1,}", " ");

            plainText = plainText.replaceAll(" {1,}", "\n");

            plainText = plainText.replaceAll("(?<![0-9])\\.", "\n$0\n\n");

            plainText = plainText.replaceAll(" [!?:] ", "$0\n");

            jCas.createView(NERReader.CONLL_VIEW);
            jCas.getView(NERReader.CONLL_VIEW).setDocumentText(plainText);
        }
        catch (CASException e) {
            throw new AnalysisEngineProcessException(e);
        }
    }
}
```

B.2 NETransformer

```
1  public class NETransformer extends JCasAnnotator_ImplBase {
2      private Logger logger;
3
4      public static final String FULL_TEXT = "fullText";
5      @ConfigurationParameter(name = FULL_TEXT, mandatory = true)
6      private static String fullText = null;
7
8      public static String LEXIA_TYPES_VIEW = "lexiaTypesView";
9
10     @Override
11     public void initialize (UimaContext context) throws ResourceInitializationException
           {
12         super. initialize (context);
13         logger = context.getLogger();
14         PersonPatterns.clearPattern();
15         LocationPatterns.clearPattern();
16         OrganisationPatterns.clearPattern();
17         OtherPatterns.clearPattern();
18     }
19
20     @Override
21     public void process(JCas jCas) throws AnalysisEngineProcessException {
22         try {
23             jCas.createView(LEXIA_TYPES_VIEW);
24             jCas.getView(LEXIA_TYPES_VIEW).setDocumentLanguage("de");
25             jCas.getView(LEXIA_TYPES_VIEW).setDocumentText(fullText);
26         }
27         catch(CASException e) {
28             e.printStackTrace();
29         }
30         FSIterator annotationIterator = jCas.getAnnotationIndex().iterator();
31         boolean entityStarted = false;
32         String currentType = null;
33         String currentPattern = null;
34         while(annotationIterator.hasNext()) {
35             Annotation annotation = (Annotation) annotationIterator.next();
36             String type = annotation.getType().getName();
37             if (type.equals("de.tudarmstadt.ukp.dkpro.core.api.ner.type.NamedEntity"))
                   {
38                 List<Feature> features = annotation.getType().getFeatures();
```

```
39                  String entityType =
                        annotation.getFeatureValueAsString(features.get(features.size() -
                        1));
40                  System.out.println("ANNO: " + annotation.getCoveredText() + " : " +
                        entityType);
41
42                  if (!entityStarted) {
43                      switch(entityType) {
44                          case "O":
45                              break;
46                          default:
47                              currentType = entityType;
48                              currentPattern = annotation.getCoveredText();
49                              entityStarted = true;
50                              break;
51                      }
52                  }
53                  else {
54                      switch(entityType) {
55                          case "O":
56                              switch(currentType) {
57                                  case "B-PER":
58                                      if (!PersonPatterns.getPersonPattern().
59                                        contains(currentPattern)) {
60                                          PersonPatterns.
61                                          addPersonPattern(currentPattern);
62                                      }
63                                      break;
64                                  case "B-LOC":
65                                      if (!LocationPatterns.getLocationPattern().
66                                        contains(currentPattern)) {
67                                          LocationPatterns.
68                                          addLocationPattern(currentPattern);
69                                      }
70                                      break;
71                                  case "B-ORG":
72                                      if (!OrganisationPatterns.getOrganisationPattern().
73                                        contains(currentPattern)) {
74                                          OrganisationPatterns.
75                                          addOrganisationPattern(currentPattern);
76                                      }
77                                      break;
78                                  case "B-OTH":
79                                      if (!OtherPatterns.getOtherPattern().
80                                        contains(currentPattern)) {
```

```
81                                          OtherPatterns.addOtherPattern(currentPattern);
82                                      }
83                                      break;
84                                  }
85                                  currentPattern = null;
86                                  currentType = null;
87                                  entityStarted = false;
88                                  break;
89                              default:
90                                  if (annotation.getCoveredText().equals(".")) {
91                                      currentPattern = currentPattern +
                                            annotation.getCoveredText();
92                                  }
93                                  else {
94                                      currentPattern = currentPattern + " " +
                                            annotation.getCoveredText();
95                                  }
96                                  break;
97                          }
98                      }
99                  }
100             }
101         }
102 }
```

B.3 DBPediaNETransformer

```
1  public class DBPediaNETransformer extends JCasAnnotator_ImplBase {
2      private Logger logger;
3
4      public static final String FULL_TEXT = "fullText";
5      @ConfigurationParameter(name = FULL_TEXT, mandatory = true)
6      private static String fullText = null;
7
8      public static String LEXIA_TYPES_VIEW = "lexiaTypesView";
9
10     @Override
11     public void initialize (UimaContext context) throws ResourceInitializationException
           {
12         super.initialize (context);
13         logger = context.getLogger();
14         PersonPatterns.clearPattern();
15         LocationPatterns.clearPattern();
16         OrganisationPatterns.clearPattern();
17         OtherPatterns.clearPattern();
18     }
19
20     @Override
21     public void process(JCas jCas) throws AnalysisEngineProcessException {
22         try {
23             jCas.createView(LEXIA_TYPES_VIEW);
24             jCas.getView(LEXIA_TYPES_VIEW).setDocumentLanguage("de");
25             jCas.getView(LEXIA_TYPES_VIEW).setDocumentText(fullText);
26         }
27         catch(CASException e) {
28             e.printStackTrace();
29         }
30         FSIterator annotationIterator = jCas.getAnnotationIndex().iterator();
31         while(annotationIterator.hasNext()) {
32             Annotation annotation = (Annotation) annotationIterator.next();
33             String type = annotation.getType().getName();
34             if (type.equals("org.dbpedia.spotlight.uima.types.JCasResource")) {
35                 List<Feature> features = annotation.getType().getFeatures();
36                 String entityType =
                       annotation.getFeatureValueAsString(features.get(features.size() -
                       2));
37                 String precedingText = jCas.getDocumentText().substring(0,
                       annotation.getBegin());
```

```
38
39                  int lastIndex = 0;
40                  int count = 0;
41                  while(lastIndex != -1) {
42                      lastIndex = precedingText.indexOf("\r", lastIndex);
43                      if (lastIndex != -1) {
44                          ++count;
45                          lastIndex += 2;
46                      }
47                  }
48                  String currentPattern;
49                  if (count > 0) {
50                      currentPattern =
                            jCas.getDocumentText().substring(annotation.getBegin() +
                            count, annotation.getEnd() + count);
51                  }
52                  else {
53                      currentPattern = annotation.getCoveredText();
54                  }
55                  if (entityType.contains("Place")) {
56                      if (!LocationPatterns.getLocationPattern().contains(currentPattern))
                            {
57                          LocationPatterns.addLocationPattern(currentPattern);
58                      }
59                  }
60                  else if (entityType.contains("Person")) {
61                      if (!PersonPatterns.getPersonPattern().contains(currentPattern)) {
62                          PersonPatterns.addPersonPattern(currentPattern);
63                      }
64                  }
65                  else if (entityType.contains("Organisation") ||
                        entityType.contains("Organization")) {
66                      if (!OrganisationPatterns.getOrganisationPattern().
67                          contains(currentPattern)) {
68                            OrganisationPatterns.addOrganisationPattern(currentPattern);
69                      }
70                  }
71                  else {
72                      if (!OtherPatterns.getOtherPattern().contains(currentPattern)) {
73                          OtherPatterns.addOtherPattern(currentPattern);
74                      }
75                  }
76              }
77          }
78      }
79  }
```

C Interviews

This appendix contains the interview guideline, as well as the notes of the performed interviews. However, the interviews are anonymised.

C.1 Interview Guideline

Interview Guideline – Semi-structured Expert Interivews

Interviewed are specialized lawyers, but also non jurists such as legal experts.

One interviews shall not exceed 30 minutes, including an optional demonstration.

In general, it shall be demonstrated to the lawyer, that the lawyer's work is already really good and appreciated. The purpose of this interview is to understand the lawyer's process as well as the information demand. Eventually, the lawyer shall be supported by our research.

Objectives of the Interviews:

- **Was sind Informationsbedarfe für einen Anwalt/Contract-Manager**
- **What are the information demands of a lawyer or legal expert?**
  - **When working on contracts?**
  - **When working on huge corpora?**

Structure:

Person:

1. Which role do you execute in your company?
2. How many years of professional work experience can you deport?
3. In which legal domain do you act?
4. Do you have a certain IT affinity?

Data:

1. With which documents do you work daily? To which kind of contracts or judgments belong these documents mainly?
2. Can you estimate your daily effort on inspecting and reworking these documents?
3. What are the crucial information you want to learn? What are the interrogations you bring on the documents?
4. Can you imagine an easier and more efficient way of managing your documents? To be more precise, according to your experience, do you see any potentials in your daily routine?

Extraction of relevant Information/NER:

1. Do you already utilize tools to manage your documents?
   a. If so: For what do these tools aim? Is it just a simple document storage or are those tools able to support you actively, for instance extracting interesting information?
   b. If not: What is your reasoning for not using supporting tools?
2. Would the extraction of named entities (extraction of information like persons, organisations, locations, dates, monetary values, and references) be beneficial to you? An incrasing number of start-ups offering such solutions arise in the USA and UK. Would it be imaginable that these tools support you in your daily routines?

   a. If not: What would be beneficial? Which data or information shall be extracted?
3. Would a structured depiction of your documents be beneficial to you? How shall such a view look like?
   a. Relevant meta information (Contracting parties, duration, contractual item, contractual volume, etc.) as a table?
   b. Rights and obligations?

Optional Demonstration:

1. Simple Example
   a. R&D Template of TUM
   b. Template was filled
   c. We show the extraction of named entities

C.2 Interview 1

Interview Guideline – Semi-structured Expert Interivews

Person:

1. Which role do you execute in your company?
    a. Own chancery
    b. Originally inherited as chancery with 5 partners and 7 lawyers
    c. Now a single chancery with 2 employees (1 lawyer and 1 secretary)
2. How many years of professional work experience can you deport?
    a. Admission since 2005, 17 years
3. In which legal domain do you act?
    a. Employment law
    b. Penology
4. Do you have a certain IT affinity?
    a. PhD in IT law
    b. Experiences in electronic signature approaches
    c. So yes for sure

Data:

1. With which documents do you work daily? To which kind of contracts or judgments belong these documents mainly?
    a. Well mix
        i. Contracts are always included
        ii. Laws must be regarded all the time
        iii. Judgments crucial as well
2. Can you estimate your daily effort on inspecting and reworking these documents?
    a. 20 minutes emails
    b. Creation of writ is more time consuming than reading
    c. Also print and read on paper base, higher concentration
3. What are the crucial information you want to learn? What are the interrogations you bring on the documents?
    a. Specific for the case
    b. Cannot be generalized
4. Can you imagine an easier and more efficient way of managing your documents? To be more precise, according to your experience, do you see any potentials in your daily routine?
    a. Digitalization can help a lot
    b. However, haptic is missed then
    c. Often a paper file must be kept
    d. Having 2-3 systems in parallel is not appealing either
    e. Once a certain document size is reached, only digital

Extraction of relevant Information/NER:

1. Do you already utilize tools to manage your documents?
    a. Officepackage, PDF tools, standard formats
2. Would the extraction of named entities (extraction of information like persons, organisations, locations, dates, monetary values, and references) be beneficial to

you? An incrasing number of start-ups offering such solutions arise in the USA and UK. Would it be imaginable that these tools support you in your daily routines?
    a. It is always beneficial
    b. Important: The context of the information must be brokered in a way that the compromised information ensures a faster process
    c. The interview partner is not sure how this shall work technically though
    d. Extracting keywords to create new sorting lists would be nice as well
    e. Context headlines can help to bundle keywords
    f. If functioning, very well
3. Would a structured depiction of your documents be beneficial to you? How shall such a view look like?
    a. Contracts differ a lot, in many contracts the task of the contract creator is to disguise information
    b. Often structure is recognized pretty fast though
    c. In the most cases, a contract is structured that well, that it can be shown to a non legal expert, and the person will understand it
    d. But still, when this can be improved, it would be awesome
    e. Such a system may be more beneficial for big chanceries with a huge amount of similar cases, than for a small chancery with many individual cases

C.3 Interview 2

Interview Guideline – Semi-structured Expert Interivews

Person:

1. Which role do you execute in your company?
    a. Lawyer at Freshfields
2. How many years of professional work experience can you deport?
    a. 1 year and 9 months
3. In which legal domain do you act?
    a. M&A (80%), Corporate (20%)
4. Do you have a certain IT affinity?
    a. Yes it is existing

Data:

1. With which documents do you work daily? To which kind of contracts or judgments belong these documents mainly?
    a. 80% Contracts, and 70% based on a template
2. Can you estimate your daily effort on inspecting and reworking these documents?
    a. 15-20% of the working hours
3. What are the crucial information you want to learn? What are the interrogations you bring on the documents?
    a. Not applicable, because the reason for looking into a contract is varying a lot
    b. Each clause can be relevant for a given case
4. Can you imagine an easier and more efficient way of managing your documents? To be more precise, according to your experience, do you see any potentials in your daily routine?
    a. Drafting can be simplified a lot
    b. Recognition of definition chains along with automatic adaption

Extraction of relevant Information/NER:

1. Do you already utilize tools to manage your documents?
    a. No tools
    b. Prefer to read printed versions
    c. At the beginning of digitalization, innovation center in London just for such purposes was just established
    d. But software such as Leverton, Kira and HotDocs in use
2. Would the extraction of named entities (extraction of information like persons, organisations, locations, dates, monetary values, and references) be beneficial to you? An incrasing number of start-ups offering such solutions arise in the USA and UK. Would it be imaginable that these tools support you in your daily routines?
    a. This would be totally beneficial
3. Would a structured depiction of your documents be beneficial to you? How shall such a view look like?
    a. Also very helpful
    b. Excel sheets, tabular form
    c. Hard to define in advance what is necessary
    d. Information demand cannot be determined easily

e. Simple extractions like agreement parties is rather easy to read without support
f. Complex structures
    i. change of control clauses
    ii. Question of liability
    iii. Representatives compensation

Optional Demonstration:

1. The interview partner liked what he has seen
2. Pretty good tool
3. Nice visualisations
4. Not yet suitable for a chancery though

C.4 Interview 3

Interview Guideline – Semi-structured Expert Interivews

Person:

1. Which role do you execute in your company?
    a. TUM
    b. Legal department
    c. Creation of contracts, negotiation of contracts
    d. Answering legal issues
2. How many years of professional work experience can you deport?
    a. Full jurist, but no admitted lawyer
    b. 4 years from September
3. In which legal domain do you act?
    a. Includes a variety of domains, such as contract law or industrial property protection
4. Do you have a certain IT affinity?
    a. Yes, it is existing

Data:

1. With which documents do you work daily? To which kind of contracts or judgments belong these documents mainly?
    a. Just contracts
    b. R&D contracts for cooperations
2. Can you estimate your daily effort on inspecting and reworking these documents?
    a. Based on 8 working hours a day, it is about 4 hours, the rest of the time is administrative tasks
3. What are the crucial information you want to learn? What are the interrogations you bring on the documents?
    a. Depends on the specific case
    b. However, standardly dispensation of privileges, distortion of privileges and liability questions
    c. Also compliance with the law
    d. But again, very difficult to generalize
4. Can you imagine an easier and more efficient way of managing your documents? To be more precise, according to your experience, do you see any potentials in your daily routine?
    a. Just better structures
    b. The interview partner only had in mind simple things such as naming of files

Extraction of relevant Information/NER:

1. Do you already utilize tools to manage your documents?
    a. Yes
    b. Shared network, storing all documents
    c. Contract drafts also shared in that network
    d. Also hard copies in use
2. Would the extraction of named entities (extraction of information like persons, organisations, locations, dates, monetary values, and references) be beneficial to

you? An incrasing number of start-ups offering such solutions arise in the USA and UK. Would it be imaginable that these tools support you in your daily routines?
    a. Yes would be, but again depend son the information demand and the recognized entities
3. Would a structured depiction of your documents be beneficial to you? How shall such a view look like?
    a. Simple meta information such as size of a rental object would be already pretty benefical
    b. Simple standards as contracting parties not though
    c. Things which can occur in a contract at every possible spot are subject to be detected
    d. Obligations and duties as well
    e. But important to keep the context always in mind, otherwise some information may be useless in a specific case
    f. Different wording of lawyers makes it difficult as well
    g. Terminology issues