TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik XIX

Semantic Analysis and Computational Modeling of Legal Documents

Bernhard Ernst Waltl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Martin Bichler
Prüfer der Dissertation:	
	1. Prof. Dr. Florian Matthes
	2. Prof. Kevin D. Ashley, PhD, University of Pittsburgh

Die Dissertation wurde am 16.04.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 19.06.2018 angenommen.

Zusammenfassung

Die Arbeit mit Rechtstexten folgt dem Paradigma der Wissensarbeit und kann als daten-, wissens-, und zeitintensiv beschrieben werden. Die Analyse von Information in Hinblick auf rechtliche Relevanz wird unzureichend von Technologie unterstützt. Dies ist kontraintuitiv, da relevante Information digital vorliegt, Algorithmen zur Verarbeitung von natürlichsprachlichem Text immer präziser und Infrastrukturen zunehmend leistungsfähiger werden.

Das Ziel dieser Arbeit ist die Identifikation der Potentiale von Technologie, die bei der inhaltlichen Analyse, Strukturierung und Formalisierung von rechtlich relevanter Information unterstützen kann. Obwohl es bereits mehrere, in der wissenschaftlichen Literatur auch beschriebenen Ansätze in diesem Bereich gibt, mangelt es diesen oftmals an Generalisierbarkeit, Wiederverwendbarkeit und der Anwendbarkeit für die deutsche Rechtsordnung.

Die Arbeit stellt ein methodisches und konzeptuelles Framework vor, das auf etablierten Methoden der Rechtstheorie aufbaut und Softwareunterstützung zur Analyse von Rechtstexten bietet. Dabei werden zwei interdisziplinäre Referenzprozesse vorgestellt: i) softwaregestützte semantische Analyse von rechtlichen Dokumenten und ii) softwaregestützte Analyse und Interpretation von Gesetzen. Das Framework basiert auf einer leistungsfähigen Softwarearchitektur, u.a. Apache UIMA und Apache Spark, und interagiert mit einem meta-model-basierten Informationssystem. Die Verfahren zur Erkennung und Extraktion von semantischen Entitäten wurden unter Einbeziehung von regelbasiertem und aktivem maschinellem Lernen (active machine learning) entwickelt. Das System verfügt über Komponenten zur Formalisierung von rechtlichen Entscheidungsstrukturen in ausführbaren model-basierten Repräsentationen in denen die Ergebnisse der Interpretation modelliert und analysiert werden können.

Die Anwendbarkeit und Performanz des Prototyps wird im Rahmen eines Industrieprojekts gezeigt, in dem automatisch Information aus einem Korpus von ca. 130.000 Dokumenten aus dem deutschen Steuerrecht, etwa Gesetze, Urteile, und Aufsätze, extrahiert wird ($F_1 = 0.92$). Die Klassifizierung von Rechtsnormen wird mit einem regel-basierten Verfahren ($F_1 = 0.78$) und mit maschinellem Lernen durchgeführt ($F_1 = 0.73$). Es wird außerdem nachgewiesen, dass aktives maschinelles Lernen dem herkömmlichen (supervised) maschinellen Lernen beim Klassifizieren von rechtlichen Normen in Gesetzen überlegen ist. Zusätzlich zeigen zwei Machbarkeitsstudien die Anwendbarkeit der Formalisierung zur Berechnung des Kindergelds und der Fristen der ordentlichen Kündigung in Mietwohnungen nach deutschem Recht.

Abstract

The work of legal scientists and legal practitioners follows the paradigm of knowledge work that is intensive in data, knowledge, and time. The analysis of information with regard to its legal relevancy still lacks the support of technological innovation. This is counter-intuitive, since the data is digitally available, algorithms for natural language processing are becoming increasingly accurate, and the available infrastructure is powerful.

The main research goal of this thesis is the identification of potentials for technology to support the semantic analysis, structuring, and formalization of legally relevant information. Although several attempts have already been made and described in scientific literature, most of the proposed solutions lack of generalizability, re-usability, and applicability — at least for the German domain.

This thesis introduces a methodological and conceptual framework, incorporating established theories from legal theory, to provide software-support for legal scientists and practitioners. It describes two interdisciplinary reference processes: i) for the software supported semantic analysis of legal documents and ii) for software support during the analysis and interpretation of statutory texts, e.g., German laws. The framework incorporates state-of-the-art software architecture, i.e., Apache UIMA and Apache Spark, and interacts with a meta model-based information system. Rule-based and active machine learning-based components were implemented to classify and extract semantic types of norms from statutory documents. This was the foundation to develop a formal calculus, i.e., model-based reasoning, and software components for decision support functionality, representing the interpretation of legal norms and enabling the computational reasoning.

The applicability and performance of the prototype is shown within an industry project with the objective to automatically extract information from a corpus of approximately 130,000 documents from the German tax law ($F_1 = 0.92$), e.g., statutes, judgments, and articles. The classification of legal norms is executed using rule-based information extraction ($F_1 = 0.78$) and using machine learning ($F_1 = 0.73$). We also show that active machine learning is superior to classical supervised machine learning in classifying legal norms in statutory texts. In addition, we performed two proof-of-concepts in formalizing the child benefit claim according to the German tax income act and the termination period of the German tenancy law.

Acknowledgment

I would like to express my special appreciation and thanks to my supervisor Prof. Dr. Florian Matthes for taking the risk of guiding me in this exciting field of research. I want to thank him for showing me research opportunities, paths, and challenges; and for allowing me to grow as an academic researcher. I further want to express my sincere gratitude to Prof. Kevin D. Ashley for joining the doctoral committee and helping me to orientate myself in this complex field. Just like Kevin inspired so many generations of young researchers, he inspired me.

This interdisciplinary research would not have been possible without the great support from Prof. Dr. Hans Christoph Grigoleit and Konrad Heßler. Prof. Grigoleit's continuous enthusiasm for the legal science fascinated me from the first day on. It was a great pleasure and I am more than grateful to have a remarkable mentor like him. In numerous discussions, Konrad's acumen and ability to understand ideas, find weaknesses, and constructively provide feedback significantly influenced my perspective on the legal science.

The sebis chair has been an excellent environment for my research. Some special thanks go to every colleague who directly or indirectly contributed to this thesis, especially Dr. Thomas Reschenhofer, Dr. Alexander Schneider, Ingo Glaser, Ulrich Gallersdörfer, Jörg Landthaler, Elena Scepankova, and Marin Zec.

I would also like to thank the students who made this research possible and forced me to formulate and rethink my too complex ideas. Special thanks go to Georg Bonczek, Johannes Muhr, Thomas Grass, Dominik Oppmann, Philipp Pickel, Patrick Ruoff, Tobias Waltl, Sirma Gjorgievska, and Daniel Jorde.

Finally, I want to thank my family for their support. I am most grateful to my parents, Michaela and Ernst, who always motivated, supported, and encouraged me to develop and continuously pursue my manifold passions. The same holds for my sister, Magdalena, and my brother, Michael, who have influenced my character in a most positive way since the earliest years of my life. I cannot express how much I owe to my patient and loving partner Isabel. Last, but not least I want to thank Daniel and Ursula for unconditionally being there whenever I got stuck — I wish I could be the son, brother, partner, and friend you all deserve.

Garching bei München, September 3, 2018

Bernhard Waltl

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The usage of information technology within the legal domain is highly appealing and has been in the center of academic research since many decades. The reasons are manifold, but are rooted in the fact that working with legal texts, such as laws, statutes, judgments, and contracts, can be considered as classical knowledge work according to Di Ciccio et al. (2015). These tasks are

- knowledge-intensive,
- data-intensive, and
- time-intensive.

Consequently, software support can make a huge impact and has been a main objective of academic and industrial research since the early beginnings of the interdisciplinary field of legal informatics. The attempts of providing adequate support were as diverse as the tasks of legal knowledge workers are and covered a very broad spectrum, such as formalizing legal rules for civil and common law jurisdictions to enable computational reasoning, structuring legal knowledge in formal ontologies, providing alternative representations and legal visualizations for legal documents, or extracting information from and semantically annotation of legal documents, i.e., legal text analytics. An excellent overview of the work form the last 25 years is provided in the overview paper from Bench-Capon et al. (2012).

Advances in computer science, software engineering, and information systems research have, in a certain way, always influenced and positively contributed to legal informatics. For example the semantic web, which was originally proposed by Berners-Lee et al. (2001), had a significant impact on the development of research communities in legal informatics, such as the International Association for Artificial Intelligence and Law (IAAIL). Their approaches investigated how to structure legal knowledge in an ontological format, thus making it accessible for knowledge representation and for computational legal reasoning (see Sartor et al. 2011a,b; Casellas 2011).

Based on the literature that summarizes the trends in legal informatics (e.g., Ashley 2017; Bench-Capon et al. 2012), it becomes evident that the most recent advances and trends in computer science, especially in information retrieval, machine learning, and artificial intelligence, influence the field of legal informatics. This trend can also be observed in recent publications: in the works on extraction of legally relevant concepts from textual legal documents; e.g., to improve search results in legal information databases (Grabmair et al. 2015), to classify legal sentences and norms (Maat et al. 2010; Maat and Winkels 2010), or to structure legal contracts(Chalkidis et al. 2017); or in work on the use of artificially intelligent application in various scenarios; e.g., predictive analytics (Nay 2017), or case-based reasoning using qualitative and quantitative information (Grabmair 2016).

Recent publications motivate the usage and integration of information technology within the legal domain for two complementary tasks:

1. **Legal text analytics** as a sub-domain of data science and text mining (see Ashley 2017; Manning and Schütze 1999; Jurafsky and Martin 2014).
2. **Computational models of legal reasoning** as a sub-domain of artificial intelligence (see Ashley 2017; Russell and Norvig 2009; Bench-Capon et al. 2012).

Throughout the last years, the field of data and text mining, including Natural Language Processing, has made significant progress, which increases the attractiveness of its usage within the legal domain, in the so-called legal text analytics. This is due to several reasons, and the following three are particularly important:

1. **Availability of (digital) data:** Throughout the legal domain, there is a trend towards open data initiatives. Legislators and jurisdictions all over the world publish more and more legally relevant information, such as laws, statutes, judgments, amendments, articles, etc., in a digital and machine-readable format. It is foreseeable that most of the documents will be published in a native digital format, i.e., not as scans, and image or character recognition are required to extract the content of a document.
2. **Accuracy and efficiency of algorithms:** Basic algorithms performing linguistic operations on unstructured information, e.g., text, become increasingly accurate. Complex and computationally expensive operations, such as usage of machine learning algorithms, are efficiently implemented. Their execution is no longer limited to powerful processing units but is possible in web applications or even on mobile devices.
3. **Capabilities of the infrastructure:** In addition to more efficient algorithms, which have become more efficient, the physical infrastructure executing them has also improved. Resources for handling the documents, such as memory and processing time, is comparably cheap and can be consumed flexibly in cloud services and virtual machines. This lowers the barrier of performing complex (legal) data science tasks, since the expenses for deploying and running these services has drastically decreased.

Based on the textual analysis and interpretation of legal texts, algorithms supporting the formalization of decision structures and the usage of artificial intelligence to "break down a complex human intellectual task" (see Ashley 2017, p. 4) are highly attractive and many different at-

tempts to formalize legal decision structures have been made (a good overview of different attempts can be found in Bench-Capon et al. (2012)).

1.1. Problem Description

The general research objective of this thesis is to contribute a framework tailored to the German legislation and jurisdiction to leverage the potential of legal text analytics and computational modeling of legal reasoning. The objective is to conceptualize and prototypically implement a collaborative environment

1. to perform state-of-the-art legal text analytics,
2. to annotate legal texts to connect them with corresponding reasoning structures, and
3. to computationally reason on interpreted legal norms.

Current implementations of text mining environments lack an adaption to the German legal system (see Ide and Pustejovsky (2017)). This lack becomes apparent in the design and conceptualization of the software frameworks regarding important software technical aspects, such as the data model, the annotation framework, the type system and the modularity of the software components. The accuracy and the efficiency of existing algorithms has continuously been improved during the last decades. However, there is *no free lunch in data science*[1]. Machine learning models and algorithms need to be adapted for a particular domain. This includes preprocessing tasks, such as splitting of sections within legal documents, but also tasks during the processing, such as defining the semantic entities and training the linguistic models accordingly. Different tasks in legal data analytics have already been attempted, such as the extraction of contract elements by Chalkidis et al. (2017), or the classification of legal norms by Maat et al. (2010); Maat and Winkels (2010) within a Dutch document corpus. However, these attempts did not focus on the provision of a standardized framework that would allow for reusing components, trained models, and patterns. This can be considered as a drawback within the legal informatics community, as no generally accepted standard for processing and annotating of legal documents has emerged.

A main goal during the analysis of legal documents is to support subsequent reasoning processes. Legal reasoning has many different notions and concepts and is also examined with respect to the different jurisdictions, e.g., common and civil law. An extensive overview of the different approaches is provided in Bench-Capon et al. (2012), where 50 papers of AI and law research are summarized. A main issue is isomorphism (see also Section 4.1), which deals with the challenge of ensuring the semantical equivalence between the legal text and the decision structures. This work tackles this issue by developing a concept and implementation to connect the text with the computational model using annotations. These could either be created automatically, by text analytics components, or manually .

Finally, based on the insights on meta-model-based knowledge engineering and domain specific languages, a framework for computational reasoning is developed to model interpreted legal

[1] An detailed discussion on the challenges and limitations was provided by Flach (2012).

norms and performing legal reasoning. The concept relies on the usage of ontological components, e.g., types, attributes, and relations; and overcomes limitations of description logics, which are normally used in ontological reasoning, by adding operations for arithmetical reasoning. As mentioned above, each ontological component can be associated with corresponding text phrases using annotations.

1.2. Research Questions

The key objective of this thesis can be derived from the problem statement and formulated as specific research questions. The main contribution is intended to narrow the gap between the relevant and most recent research directions from legal informatics and state-of-the-art software engineering, which leads to the following research hypothesis:

> **Research hypothesis:** A collaborative software environment can support the semantic annotation of legal documents, using text analytics components to design and formalize computational models within ontological decision structures.

The hypothesis can be divided into different research questions allowing a structured approach to address the overall objective. The following research questions are addressed and answered within this thesis:

> **Research question 1:** What is the state-of-the-art in software-supported analysis of textual documents in the legal domain focusing on legislative and judicial texts?

The state-of-the-art in software-supported analysis is elaborated with an overview of relevant literature considering recent publications and projects in the field of legal informatics. A differentiated analysis of the legal systems, such as civil and common law, needs to be taken into account. The focus of the literature overview is to show the potentials of semantic analysis of legal documents with regard to different use cases and scenarios (see Section 2.1). In addition, an investigation of implemented software frameworks and platforms was performed and evaluated in Section 3.

> **Research question 2:** What are the methods and tools to formalize ontological decision structures emerging from statutory texts?

In order to investigate the potentials of representing interpreted decision structures in formalized and computable models, the most common methods and tools need to be investigated (see Section 2.3). The main focus is to understand the potentials of formalizing statutory texts, such as laws and statutes among others. Not only the tools and formal calculus but also the processes of how these decision structures can be derived, i.e., interpretation methods, are taken into account .

> **Research question 3:** What could a reference process, considering activities, roles, artifacts, and software tools, for the software-supported semantic analysis of legal documents look like?

Based on the consideration of the state-of-the-art for the semantic analysis of legal documents, a reference process is defined (see Section 3.1). The process is subdivided into different activities. In addition to the activities, central roles within the interdisciplinary reference process are identified. The reference process also takes into account artifacts, e.g., documents, type systems, etc. The activities are the base line for the identification, conceptualization and implementation of the software services and tools.

> **Research question 4:** What are the design principles and components of a software architecture enabling a collaborative environment for information extraction processes from legal documents?

As modern software engineering heavily relies on the principle of reusing software and components, a state-of-the-art software has to be designed to support this principle is supported. The architecture and design, as proposed in Section 3.4 and Section 3.5, has been developed to take this into account. Components are loosely coupled and well-defined interfaces ensure the modularity of the overall system. This needs to be considered for the management of legal documents, including the data model and central data storage, and their semantic processing; including a generic but robust, e.g., a thread-safe pipeline architecture.

Based on existing implementations for software architectures to perform information extraction processes, the implications for the design of a collaborative legal data science environment are investigated. The question focuses on the design of a software architecture to manage and semantically analyze legal documents. In addition, the collaborative part is considered by the system, as the processing of documents and modeling of decision structures can be performed simultaneously by different users.

> **Research question 5:** What are the elements of a reference process with the aim of formalizing statutory texts into computational decision structures?

Similar to the reference process model for the interdisciplinary legal data science a reference process model for the analysis and interpretation of statutory texts is defined (see Section 4.1). Based on commonly accepted interpretation methods from legal theory the process model identifies different activities following the analysis of statutory texts, their interpretation, and finally their application in terms of legal decision-making. The software support for the individual activities is discussed. The subdivision into different phases allows for a better structure of the software support.

> **Research question 6:** What are the components of an ontological reasoning framework modeling the computational semantics and connecting the interpreted legal texts with the corresponding model element?

Based on the semantic analysis of legal documents and the literature on the formalization of statutory text into computational models, knowledge engineering approaches, i.e., ontological concepts, and legal reasoning are combined in Section 4. The model-based reasoning differentiates between the structuring of the legal knowledge and an executable part, so reasoning is possible based on the provided facts and on evidence (see Section 4). The information extracted from the legal text and stored as annotations can be linked to each model component to maintain the source of the interpretation from the textual documents.

In addition, the full-stack design and concept for the decision support system is developed based on the reference process (see Section 4.3). Subsequently, the user support for the analysis and inspection of the decision structures is discussed (see Section 4.4).

> **Research question 7:** Which degree of accuracy can be achieved during information extraction and norm classification in judicial and statutory texts? What are the limitations and emerging research directions?

Finally, the quality of information extraction components and the modularity of software architecture of the environment need to be assessed. For this purpose, different aspects of the system are investigated individually: the suitability of the architecture to support editorial staff by extracting particular information from tax law documents (see Section 6.2), the classification of legal norms with rule-based and active machine learning approaches (see Sections 6.3 and 6.4), and the formalization of one particular decision structure in Germany's tenancy law (see Section 6.5). This assessment facilitates a detailed discussion about the potentials and limitations of the semantic analysis and formalization, based on which subsequent research directions can be derived.

Against the background of this explicit formulation of the research questions, the research design can be chosen in order to adequately address and approach the overall research hypothesis can be selected.

1.3. Epistemological Position and Research Design

An adapted Information Systems Research (ISR) research framework (originally proposed by Hevner et al. (2004)) has been modified for the thesis at hand. Figure 1.1 depicts the research paradigms and consists of the following main elements:

- **Environment**: The phenomenon of interest is embedded into the problem space, which is defined as the environment. For the field of ISR, people, organizations, and their technologies are subsumed therein.
- **Business Needs**: The goals, tasks, and problems define the business needs. These are assessed and evaluated within the context of organizational strategies, structures, and existing business processes.
- **IS Research**: Design science aims at building and evaluating an artifact to meet the previously determined business needs. The goal of design science is utility. The assessment of the artifacts results in an identification of weaknesses of the artifact and in the need to refine and reassess it.
- **Knowledge Base**: The knowledge base provides the raw materials from and through which ISR is accomplished. The knowledge base consists of foundations and methodologies, prior IS research and results from theories, frameworks, models, and methods used in the develop phase.

- **Applicable Knowledge**: The applicable knowledge emerges from the knowledge base. It influences the design and implementation of the artifact, and its assessment.

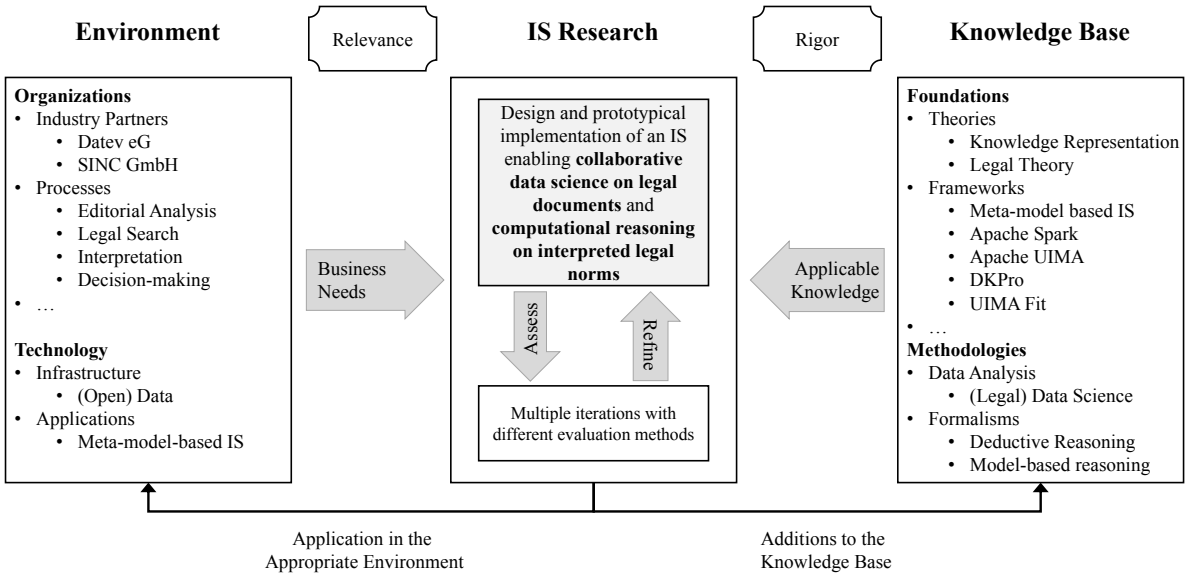


Figure 1.1.: Adapted information systems research framework proposed by Hevner et al. (2004).

According to Hevner et al. (2004), "design science is inherently a problem-solving process". They derived seven fundamental principles, i.e., guidelines, of design-science research that structure the knowledge and understanding of a design problem and its solution. The design-science process heavily relies on designing, building, and applying an artifact. The following considerations discuss the applicability of design-science for this thesis to address the problem statement and research questions stated above.

Guideline 1 "Design as an Artifact": The most important entity within the research process is the artifact which needs to be built. According to (Hevner et al., 2004, p. 82), "the result of design-science research in IS is, by definition, a purposeful IT artifact created to address an [...] organizational problem. It must be described effectively, enabling its implementation and application in an appropriate domain.". In the scope of this thesis, the artifact is a collaborative web application that enables its users to apply state-of-the-art text analytics tools to semantically analyze legal documents. This analysis should ultimately support the interpretation and creation of computable representations of legal decision structures.

Guideline 2 "Problem Relevance": Supporting the research problem by proving its relevancy for a business problem is a main motivation of design-science: "[...] to develop technology-based solutions to important and relevant business problems." (Hevner et al., 2004, p. 83). The problem of the textual analysis of legal documents is commonly accepted in the field of legal informatics as discussed by Ashley (2017) and in Bench-Capon et al. (2012). In addition, the commercial relevancy is discussed, e.g., by Susskind (2013).

Guideline 3 "Design Evaluation": Once a design artifact has been built, its utility and

efficacy must rigorously be shown via established evaluation methods. The methods that can be used are manifold, but need to be in line with the commonly accepted methodologies, such as case studies, or analytical approaches. The research prototype developed in this thesis was used within a case study with an industry partner (see Section 6.2), and the overall performance was demonstrated in two complementary quantitative evaluations (see Sections 6.3 and 6.4).

Guideline 4 "Research Contribution": According to Hevner et al. (2004, p. 83), effective design-science research "[...] must provide clear and verifiable contributions in the areas of the design artifact [...]". This highlights the importance of clarifying the distinction achieved through the research. The contribution made by creating and developing a new artifact needs to show an advancement regarding the epistemological increase of knowledge. The contribution of this thesis are the prototype, a reference process models for interdisciplinary legal data science (see Section 3), and a reference process model for the interpretation and formalization of statutory texts (see Section 4).

Guideline 5 "Research Rigor": A research gap was determined by analyzing related literature on the analysis of legal documents with particular focus on statutory texts for the German domain. Main challenges during the conceptualization of the framework were the design and implementation of a modular framework fostering collaborative data science while preserving the particularities of German legal documents, e.g., a data model for efficiently representing statutory texts. The computational modeling for model-based reasoning on interpreted legal norms does not only take into account existing reasoning approaches, e.g., ontological reasoning with description logic.

Guideline 6 "Design as a Search Process": The IT artifact developed has continuously been evaluated, extended, and improved. Thereby, more and more software components have been added to allow an elaborate semantic analysis of legal documents. The functionality of the overall system increased steadily, while desirable design principles were preserved, such as software modularity and loose coupling among the components. The concepts and implementations in the present thesis present can be considered to be a detailed description of the current state of the search process, on which more generate/test cycles (see Hevner et al. 2004, p. 89) can be performed.

Guideline 7 "Communication of Research": Finally, the research prototype as well as selected functionalities have to be presented to appropriate audiences, focusing on either technology or use-case scenarios. Figure 1.2 summarizes the main publications authored during the research this thesis is based on.

Based on the considerations for design-science in information systems research, the main contribution and the results of the thesis were established. The next section provides more details about the remainder of the thesis in more detail.

1.4. Outline of the Thesis

This thesis is organized by seven chapters describing the research and its contribution. Figure 1.2 shows the structure of the thesis and outlines the chapters II–VII. The figure is divided

into four different pillars: chapters, research questions, research results & artifacts, and main publications. This serves as an overview of the different parts and pieces and to show how they are interconnected and built on top of each other.

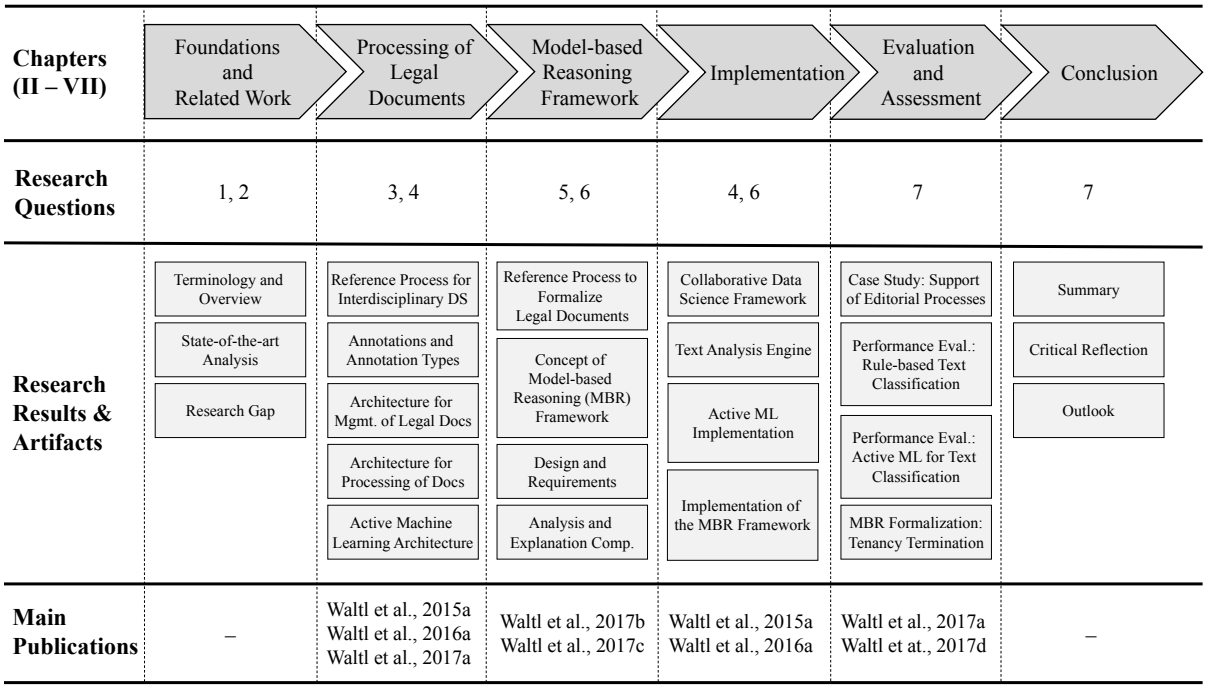


Figure 1.2.: Structure, outline, and main contribution of this thesis.

The following paragraphs briefly outline the individual chapters.

Chapter 2 "Foundations and Related Work" summarizes the main publications relevant for the research hypothesis and research questions. The main body of academic research papers has been published by the legal informatics community. This is particularly relevant, as the problem at hand is a genuinely interdisciplinary research problem. The related work section differentiates between the problem of legal text analytics and the formalization of statutory legal documents. The main contribution thereof is the clarification of terminology and the identification of a research gap.

Chapter 3 "Semantic Analysis and Annotation of Legal Documents" continues with an extensive discussion of how legal documents are processed. Thereby, an interdisciplinary reference process model is proposed which was evaluated within a case study. The role of annotations and annotation types is discussed and the relevancy for the domain of legal text analytics is explicated. The chapter also introduces the concept for managing legal documents within a collaborative web application. The main contribution is the architecture for processing legal documents in a Pipes & Filters architecture, fostering the reuse and modularity of software components. The role of different components, such as rule-based information extraction, is

discussed, and their importance for the legal domain is shown. In addition, the role of Active Machine Learning is detailed and its architectural integration is described.

Chapter 4 "Concept and Design of a Model-based Reasoning Framework" contributes the formalization of statutory texts into model-based decision structures. At the beginning it proposes a reference process model describing four different iterative activities that are performed during the analysis and interpretation of a statutory text. The process model also shows the potential of software & tool support. Based on these considerations, an example shows the limitations of current ontological reasoning structures, e.g., description logic. The main requirements for the reasoning framework are derived and the role of analysis and explanation components within legal decision support systems is discussed.

In **Chapter 5 "Implementation"**, the prototypical implementation is shown by highlighting the main parts of the concrete software. Important and illustrative parts of the implementation are discussed in detail to show the necessity of a well-structured software architecture and their contribution to the overall software artifact. The implementation chapter is organized along three different categories: legal text analytics, active machine learning, and model-based reasoning.

Finally, in **Chapter 6 "Evaluation and Assessment"**, the research artifact is evaluated. For this purpose, four different evaluation phases were conducted using three different methods: case study with an industry partner, quantitative analysis of the information retrieval components, and a proof-of-concept for the generalization of the formalization method.

In **Chapter 7 "Conclusion"**, the contributions of this thesis are summarized and critically reflected. The initially formulated research questions are reconsidered, and answers are provided. Based on these considerations, limitations are identified and open questions for future research are proposed.

CHAPTER 2

Foundations and Related Work

This chapter constitutes the foundation for the concept, design, and implementation as described in this thesis. It provides an overview of the state-of-the-art in different but related domains investigating legal text analytics and computational models of legal reasoning in the German legal system.

The chapter is divided into three main sections:

- Section 2.1 "Legal Text Analytics and Software Engineering" summarizes the main aspects of text analytics and Natural Language Processing with a particular focus on the legal domain. It takes into account concepts from the domain of computational linguistics and software engineering.
- Section 2.2 "Representing the Structure of Legal Documents" introduces the most common and widely accepted concepts to technically represent legal documents. The purpose is to provide a differentiated view on modeling the structure and the semantics of a legal document.
- Section 2.3 "Computational Models of Legal Reasoning" describes the main research results from the domain of legal reasoning into formalized models. Different aspects from a logical, computational, and practical point of view are considered. The focal point is analyzing the approaches and contributions for reasoning on formalized statutes and civil law jurisdictions.

2.1. Legal Text Analytics and Software Engineering

2.1.1. Foundations of Text Analytics

Natural Language Processing (NLP) has a long tradition and the main advances have been made the field of computational linguistics. The main objective thereby is to develop concepts and algorithms to process human (natural) speech and language. This section will introduce the main concepts and highlight the knowledge base of text analytics with a particular emphasis on processing legal texts.

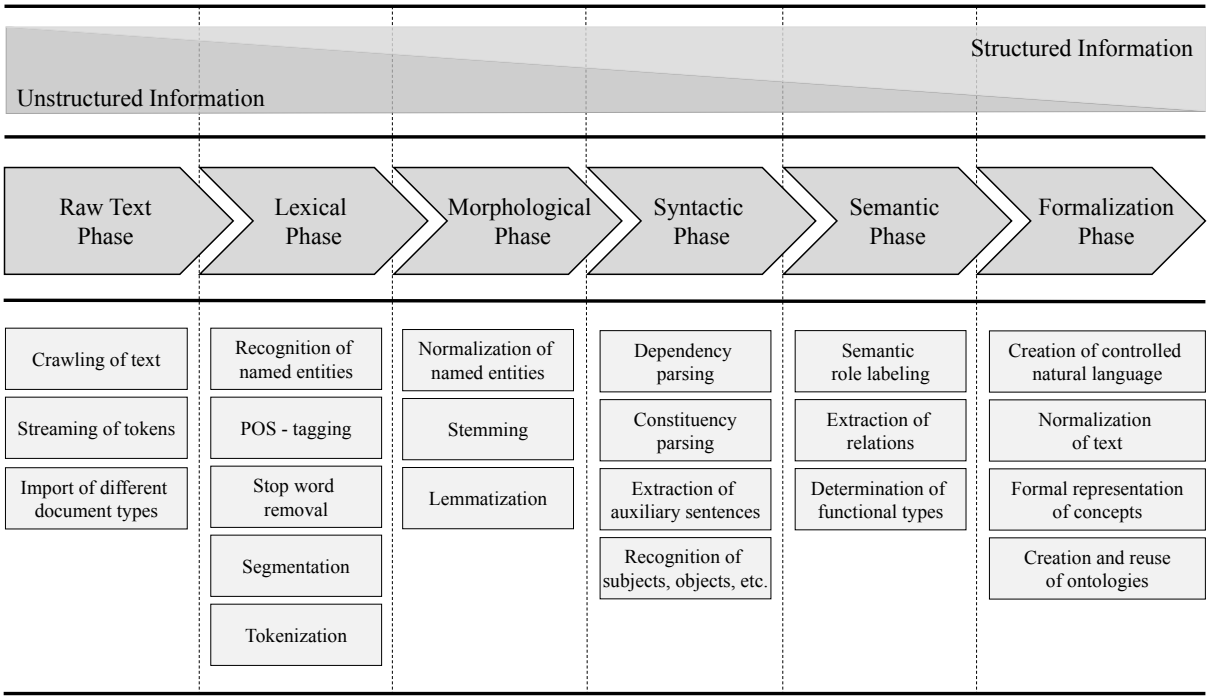


Figure 2.1.: Transition from unstructured to structured information using computational linguistics (based on Ide and Pustejovsky (2017); Jurafsky and Martin (2014); Manning and Schütze (1999); Bishop (2006)).

Based on existing literature, a main goal of NLP and text mining could be described as processing unstructured information, i.e., text, by extracting the inherent structures (Jurafsky and Martin, 2014). The ultimate goal is to formalize the unstructured information into in a computational format. Figure 2.1 presents an overview on how the transition from text to a formal specification is performed. In this process, various tasks and phases exist in which different operations can be applied to extract knowledge from the text or to assign an additional meaning, i.e., semantics, to the text. It briefly outlines the main understanding of NLP that underlies this thesis, namely the use of several procedures and techniques to subsequently provide more and more structure for initially unstructured information.

Further information about different components, how these are designed, implemented, and

trained were summarized by Jurafsky and Martin (2014) or Manning and Schütze (1999). The next sections will focus on the concepts and applications of NLP for the legal text analytics, as defined by Ashley (2017).

2.1.2. Text Mining to Extract Concepts from Legal Documents

The application of NLP within the legal domain is very appealing and numerous approaches using different technologies and specific objectives exist. The foundations discussed for this thesis particularly focuses on the usage of NLP to extract legal concepts from legislative texts to support legal document interpretation.

In essence, three main streams can be observed: legal text analytics for legislative texts (see Biagioli et al. 2005; Savelka et al. 2015), for judicial texts (see Wyner et al. 2010; Wyner and Peters 2010a; Grabmair et al. 2015; Walker et al. 2017), and for contracts (see Chalkidis et al. 2017).

Ashley (2017, p. 233) states two main roles of text analytics for the legal domain, namely capturing the role of sentences in legal documents, e.g., arguments, and supporting conceptual information retrieval in legal information systems. Having a legal document semantically analyzed and annotated with functional information could to re-rank search results, e.g., Grabmair et al. (2015), could help users and applications form arguments or hypothesis, and potentially predict the outcome of legal cases. Although Ashley (2017) clearly focalizes the analysis of legal documents for common law jurisdictions, he emphasizes the role of NLP as assistance during the extraction of information from statutory texts. In Ashley (2017, p. 260), he identifies and summarizes the potentials in the extraction and classification or norms (e.g., functional categories), texts (e.g., topics of law), or legal rules (including antecedents and consequents).

The collection of publications in Francesconi (2010) provides a good overview on the semantic processing of legal documents. The contributions cover different topics and emphasize the role of ontologies for legal knowledge representation and structuring. The collection of publications shows a variety of applications and use cases to be performed at the intersection between law and informatics. The legal text analytics contributions focus on named entity recognition and extraction of legal terms, i.e., keywords. Only one contribution explicitly investigates the analysis of statutory text, namely the classification of functional types of norms Maat and Winkels (2010), which is discussed in depth in Section 2.1.3.

Savelka et al. (2015) processed statutes with the aim to determine the relevancy of a statutory provision given a specific case. Based on two datasets, consisting of 4,022 individual provisions and 1,532 individual provisions respectively, the authors used an interactive machine learning framework to determine the binary classification (relevant vs. non-relevant). They used a support vector machine with a linear kernel and as features, unigram representations of the provisions were used. The experiment showed that interactive machine learning is superior to classical machine learning and that the classification can support during relevancy assessment. However, the overall performance reached an $F_1 = 0.50$, and $F_1 = 0.61$ respectively. Which does not seem to be too promising at a first glance, but given the complexity of the task this can be considered as a remarkable result.

In addition to the approaches to analyze statutory texts, legal informatics structures the information contained in judicial texts, i.e., judgment, cases, court decisions, etc. The publication Wyner et al. (2010) describes an attempt to automatically mine arguments and legal case factors from legal cases. Different corpora were used and the processing was done using context-free-grammars, i.e., rule-based pattern matching. Based on a set of 20 cases from the European Court of Human Rights (ECHR), the extraction could be performed with an F_1 score reaching from 0.64 to 1, depending on the argumentation type.

Also based on the analysis of judicial documents, Grabmair et al. (2015) investigated the re-ranking of search results using annotations which reflected particularly significant textual phrases in a document corpus. The gold standard consisted of 35 case documents and the textual phrases were annotated on the sentence level focusing on the argumentation in judicial decisions. They were able to show that annotation the a sentence level, as additional information to re-rank the search results, improves the performance compared to the baseline, which was the standard search function of WestlawNext[1], a common legal information provider. The sentences were classified using a combination of rule-based and supervised machine learning approach.

The research group published an approach on a similar, very convincing idea of annotating sentences in judicial texts with regard to their semantic role and functional type (see Walker et al. 2017). Therefore, ten different semantic roles have been identified, e.g., "evidence sentence or clause", "rule-based-reasoning sentence or clause", etc. Based on a document corpus of veterans' disability claims (55,713 by 2015), the computational reasoning on these claims will be supported. The main idea is to assigning the semantic types to the sentences of this corpus. The overall goal is to "make claims processes more transparent, efficient and accurate" (Walker et al., 2017). The paper can be considered as the theoretical foundation and the results of the classification are going to be created subsequently.

Little research has been done focusing on the analysis of German legal documents. The research conducted by Walter (2010) is worth to be mentioned. In his dissertation, he focused on the extraction of definitions from cases decided by the German constitutional court, which is known to be very challenging. The labeling was done on the sentence level; the training data consisted of 3,757 sentences, form which 138 defining sentences were determined manually. The overall F_1 score was very low, but slightly varied, depending on different techniques. However, it was never above 0.14, or 0.40 respectively. Walter has chosen a rule-based approach, but did not use any context-free grammar. Instead he used an implementation in SQL, respectively XPATH, which does not support the reuse of his results and research. Nonetheless, this approach focused on German legal documents, which are hardly studied in the domain of legal informatics.

Recently an approach to classify Chinese judgments was performed by Lei et al. (2017). They manually labeled 6,735 documents into 13 different categories representing the industry division to which a particular judgment was relevant, e.g., pharmaceutical, textile and garment, transportation, etc. With an experiment they compared different parameter settings and classifiers, namely naive bayes, decision trees, random forests, and a support vector machine. They showed that SVM performs best ($F_1 = 0.87$) but also requires the most time in training, i.e., more than 15h. The remaining classifiers were trained in less than 5 minutes. They show the overall problem very well, as 70,000 new judgments are indexed in the Chines online system every day. This

[1] https://next.westlaw.com/, accessed on September 3, 2018

increases the appeal to focus on this domain and jurisdiction. More research can be expected in this particular field.

An approach regarding the the analysis of contracts was performed by Chalkidis et al. (2017). They used a mixture of hand-written rules and linear classifiers to extract elements from contracts, such as contract title, contracting parties, start date, etc. They intelligently combined zoning information and reached a performance of over 0.90 in terms of F_1 in a variety of different contract elements. Although their approach showed impressing values for a large set of contract elements, some elements, such as "Value", i.e., price, salary, etc., could only be extracted with a F_1 score of 0.64. Their training data consisted of 3,500 contracts. The analysis of contracts is highly attractive, especially for commercial initiatives. However, as this thesis mainly investigates the analysis of legislative texts, no in-depth literature analysis has been performed on the research available for the analysis of contracts.

This overview on the state-of-the art in the analysis of judicial and legislative texts using methods from computational linguistics emphasizes the relevancy, but also the diverse use cases and scenarios that exist in the field. The next section will focus on the classification of legal norms within statutory texts in particular.

2.1.3. Text Mining to Classify Legal Norms

An early contribution for the classification of norms in legislative texts has been published by Biagioli et al. (2005). The authors differentiate between the "formal partition", i.e., structure, and the semantic units of a regulation. They shared the idea, that the semantic annotation "can make the [information] retrieval easier", which is still relevant today (Ashley, 2017). In order to classify legal norms, the authors distinguish eleven different types, which are assigned to paragraphs. In an experiment, where they manually labeled 582 paragraphs, they used a multi-class support vector machine. Overall, they achieved an average F_1 measure of 0.80, whereas the precision and recall values for different classes where quite diverse, scaling from $F_1 = 0.35$ for permission to $F_1 = 0.97$ for substitutions. Francesconi and Passerini (2007) evaluated the classifiers, naive bayes, and a support vector machine by determining the same eleven functional types. They showed that the support vector machine outperforms naive bayes in the classification task.

Various attempts to classify legal norms have been made using Dutch legislative texts. Maat and Winkels (2007) discern different rule types. In their model, they follow the distinction between primary and secondary rules, originally proposed by Hart and Green (2012). Whereas the primary rules describe the main sources of normative regulation, such as rights and duties, secondary rules subsume norms that regulate the management of rules, such as their applicability and transitional provisions.

Based on this model of legal norms, they implemented two different approaches to automatically classify norms in Dutch legislative texts. Maat and Winkels (2010) followed a knowledge engineering approach by extracting typical text patterns that identify the category of a given norm. The patterns were coded into regular expressions and applied to a corpus of 18 Dutch legislative texts (Income Tax Act). (Maat and Winkels, 2010, pp. 175) also discussed the prob-

lem of finding the right granularity for the classification, but finally used the sentence level. The shortest document consisted only three sentences, whereas the longest document contained 166 sentences. In total, they derived 87 patterns to classify the sentences into 15 different categories. Using this approach, they reached an average F_1 score of 0.91. They also analyzed the patterns that they originally determined and how they contributed to the overall result. Their investigation showed that out of the 87 patterns, only 44 actually triggered a classification, whereas the remaining ones where not used at all; or at least not in the document corpus that was used for testing.

Maat et al. (2010) extended the research performed with a knowledge engineering, i.e., rule-based information extraction, by applying a machine learning classifier. This makes it a very interesting and valuable contribution to research, as hardly any attempts exists to perform a structured comparison between a knowledge-engineered approach and a machine learning-based approach. The accuracy rates have again reached the high level of 0.94. They carried out various different parameter studies and showed that binary term weight, with the removal of stop words and a minimal term frequency of 2, performed best. In an informal discussion about potential errors, they identified the "skewness" as a potential source of errors. This means, that classes of norms, that hardly occur throughout legislative texts, tend to be classified less likely, compared to those classes that occur more often.

Wyner and Peters (2010b) also used a rule-based approach to extract elements from statutory texts. They selected a passage from the "US Code of Federal Regulations, US Food and Drug Administration, Department of Health and Human Services regulation" for blood banks on testing requirements for communicable disease agents in human blood, title 21, part 610, section 40. The document contains 1,777 words and is therefore relatively small. However, their approach did not concentrate on the analysis of the norms, but they did a much finer granular analysis using the Java Annotation Patterns Engine grammar. They focused on analysis with regard to deontic rules. Their analysis model included deontic modals and verbs, agents and themes, and conditional sentences with antecedents and consequences. They reached an average F_1 score of 0.79. However, many of their deontic concepts were extracted without any error ($F_1 = 1$). The potential sources of errors are discussed extensively. The main challenge is the syntactic position of subject, object, and by-phrases and the usage of active and passive sentences.

The related work above delimits the research area and summarizes the state-of-the-art in literature on semantically analyzing legal document. However, few equivalent attempts have been made in the German domain and rarely did researchers focus on the provision of a more generic framework to foster the semantic annotation of these documents, instead of having stand-alone solutions that do not allow for the reuse of training data, trained linguistics models, and rules. Furthermore, none of the approaches above fosters collaboration of data scientists over a common platform or environment.

The next section illustrates the role of software architectures for legal text analytics. Based on the different use cases and scenarios that have been implemented, the technology used will be discussed. The commonalities are highlighted, so that the importance and the feasibility of a common platform become evident.

2.1.4. Software Architectures for Legal Text Analytics

Various attempts have carried out to analyze legal documents using software components. However, as a close look at the knowledge base shows, none of the attempts focused on the provision of a more generic software framework that could serve as a base line for further approaches in legal text analytics. Instead, most research projects start with the re-implementation of a various of software components. Depending on the concrete research, these cover almost the whole spectrum of text analytics as shown in Figure 2.1.

The remarkable work on the analysis and classification of norms from laws in the Dutch legislation by Maat and Winkels (2007); Maat et al. (2010); Maat and Winkels (2010) mainly used regular expressions and stand-alone machine learning packages, e.g., Weka (see Section 3.5). They manually extracted text patterns that can be used to classify norms and provision in legal texts. These patterns were transformed into regular expressions. Admittedly, that main focus of their research was not to design and implement a software framework that could easily be expanded to new research questions. However, this would certainly foster innovation and the reuse of their implementations and patterns. This can be applied to many of the attempts within the domain of legal text analytics. Without questioning their success, their solutions lack the potential reuse, thus preventing their achievements from a wider adoption.

The approaches from Grabmair et al. (2015) and Walker et al. (2017) are interesting from the legal perspective, but also from the software technology aspect: The underlying software framework for the analysis is based on the Apache UIMA framework, which will be discussed in detail in Section 3.5. This robust and widely adopted software architecture allows for the reuse of implemented software components, and it also fosters their exchange among the scientific community. The interchange of trained models, software components, and extracted patterns, can be highly recommended, not at least due to the expensive tasks underlying the implementation.

Another framework that is widely adopted within the domain of legal text analytics is the so-called General Architecture for Text Engineering (GATE). This framework was, for example used by Wyner et al. (2010) and Wyner and Peters (2010a). It also supports the reuse of software components, which are organized within processing pipelines. General Architecture for Text Engineering will be discussed in depth in Section 3.5. It comes along with a powerful rule language (JAPE grammar) that allows the specification of complex linguistic and textual patterns.

In Section 3, the problem and challenges for text analytics frameworks in an interdisciplinary and collaborative setting will be addressed in depth. Based on the considerations from existing research, a recommendation and prototypical implementation of an Apache Unstructured Information Management Architecture (UIMA) framework will be carried out to show its applicability for legal text analytics. As recent studies in the field of computational linguistics have shown (see Ide and Pustejovsky 2017) no standard framework has emerged throughout the field. For the small domain of legal text analytics however, this seems to be a desirable goal, as the usage of a standardized format for semantic analysis would foster and integrate the approaches in semantic analysis. Throughout the community, pre-processing, and training phases would only need to be performed once, since others could rely on the progress made by the community.

2.2. Representing the Structure of Legal Documents

To harmonize the processing within a common software framework the legal documents, including legislative, judicial, and contractual texts, need to be represented by a common data structure. Different attempts to generate this common data structure have already been carried out. Overall, Extensible Markup Language (XML) seems to emerge as a de-facto standard for managing the legal documents and for structuring their content.

The most advanced standard for specifying the structure of legal documents is provided by the OASIS committee on LegalDocumentXML. It is an XML vocabulary[2] that provides a generic structure for various legal documents: "The LegalDocumentXML Specifications provide a common legal document standard for the specification of parliamentary, legislative, and judicial documents, for their interchange between institutions anywhere in the world, and for the creation of a common data and metadata that allows experience, expertise, and tools to be shared and extended by all participating peers, courts, parliaments, assemblies, congresses, and administrative branches of governments." (Palmirani et al., 2017). The standard forms a *superset* of all possible and known document formats and differentiates between different elements and attributes on a very fine granular level. Two main objectives address the interoperability of documents, and two levels of interoperability are distinguished:

- **Semantic interoperability** should ensure the understandability of the well-defined meaning of information for any entity receiving the data, i.e., a person or application.
- **Technical interoperability** should ensure that applications, systems, and interfaces are based on a shared set of technologies, languages, and technical requirements decreasing the complexity to exchange data, access data, and the reuse of competencies and tools.

The LegalDocumentXML is a generic mark-up language that can, due to the fine granular differentiation, no longer be considered as a light-weight format in formalizing the interchange format of legal documents.

In addition to the LegalDocumentXML standard of Akomo Ntoso, the CEN MetaLex standard exist, which describes itself as "an interchange format, a lowest common denominator for other standards, intended not to replace jurisdiction-specific standards and vendor-specific formats in the publications process but to impose a standardized view on legal documents for the purposes of information exchange and interoperability in the context of software development." (Boer et al., 2002). The MetaLex standard was published earlier than the LegalDocumentXML and is used by the legislators in the UK legislation[3]. The standard essentially defines the basic structure of documents and standardizes the usage of attributes and elements throughout legislative documents. Therefore, five different main requirements where considered for the implementation:

1. Schema extension
2. Adding metadata

[2] http://docs.oasis-open.org/legaldocml/akn-core/v1.0/cs01/part1-vocabulary/akn-core-v1.0-cs01-part1-vocabulary.pdf, accessed on September 3, 2018

[3] http://www.legislation.gov.uk/, accessed on September 3, 2018

3. Cross referencing
4. Constructing compound documents
5. Basic naming conventions

Currently, the implementation of this standard across different governments lags behind. National governments still use their own schemata for the (online) publishing of legal documents. The German legislator publishes laws on a federal level using a XML format that does not adhere to the existing international standards. The schema is defined as a Document Type Definition (DTD) and can publicly be retrieved[4]. The only objective that the schema pursues, is to standardize the structure of the laws that are published online. The standard was created in 2012 and defines a small set of metadata and the information on how documents can be nested.

Besides all the attempts on formalizing the structure in a commonly accepted XML schema, the challenge remains that most NLP components cannot deal with XML, but prefer plain text. This thesis will discuss this challenge and develop a light-weight data model (see Section 5.1.2), which represents documents with their meta-information and nested structure, so that it can easily be processed by NLP components.

2.3. Computational Models of Legal Reasoning

2.3.1. A Short Introduction to Legal Expert and Decision Support Systems

Legal Expert Systems (LES) are still highly attractive for academic research, but also for industrial applications and, according to Leith, these are reasons for it:"[...] making the knowledge and expertise easily replicated, readily distributed, and essentially immortal." (Leith, 2010). Legal Expert System (LES) are well-studied throughout the domain of artificial intelligence and law. Various attempts have been made, which lead to different and valuable concepts and implementations (see Bench-Capon et al. 2012). Early concepts and implementations of reasoning, e.g., logic engines, focused on the provision of systems that allow the reasoning on propositional logic. Highly tailored to the legal domain, the usage of those systems was left up to experts with knowledge in both legal sciences and computer science. The provision of those logic systems is very attractive in civil law countries, such as Germany (see Jandach 1993). Legal systems with a predominant case law, such as the United States, have maintained a stronger focus on systems supporting case based reasoning (see Ashley 2002).

The reasoning within legal expert systems, which is discussed since decades, can be subdivided into several categories, such as

- deductive reasoning (rule-based), e.g., Prakken and Sartor (2015),
- ontological reasoning (model-based), e.g., Casellas (2011); Sartor et al. (2011a),
- deontic reasoning, e.g., Jones and Sergot (1992); Sartor (2005),

[4] http://www.gesetze-im-internet.de/dtd/1.01/gii-norm.dtd, accessed on September 3, 2018

- case-based reasoning, e.g., Ashley and Rissland (1988); Modgil and Prakken (2014); Grabmair (2016),
- abductive reasoning, e.g., Walton (2014),
- defeasible reasoning, e.g., Sartor (1995),
- probabilistic reasoning, e.g., Gerathewohl (1987); Timmer et al. (2015), and
- statistical reasoning (machine learning-based), e.g., Katz (2012); Grabmair (2016).

In countries with civil law jurisdictions the use of rule-based systems is highly attractive, because there are strong arguments for the assumption that those rules can — to a large degree — reflect statutory texts (see also Prakken and Sartor 2015). Rule-based systems represent the deductive nature of laws and regulations in civil law jurisdictions.

2.3.2. Rule-based Reasoning on Laws and Statutes

Reasoning on laws and statutes using rule-based systems has been studied by Sergot et al. (1986) in formalizing the British Nationality Act (BNA). The BNA is an excellent example for the translation into a logic program, because it embodies all characteristics of statutes, namely syntactic complexity, vagueness, and references to previously enacted legislation. Their formalization was carried out in Prolog. The BNA already contained a couple of very interesting vague phrases, such as "being a good character" or "having sufficient knowledge of English" (Sergot et al., 1986, p. 370). Although the BNA was relatively self-contained (i.e., not many dependencies to other laws and statutes), the authors describe severe problems in representing the knowledge and the knowledge elicitation, which is "the most important of the central problems of artificial intelligence research" and "the critical bottleneck." (Sergot et al., 1986, p. 382). Their final implementation consisted of approximately 500 rules and was executable by any Prolog engine. Based on their experiences, they already derived three requirements for the knowledge representation, which needs to be

1. easy for both laymen and experts to understand;
2. easy to modify (e.g., to correct errors, to enhance, and to reflect changes that occur over time);
3. capable of allowing the inference procedure to interact naturally with the human user and to explain its conclusions.

Susskind (1987) summarized the different approaches that were already undertaken to design and implement LESs at that time. He described many such attempts and derived more abstract principles and implementation guidelines. Most use cases of formalizing focused on the implementation into a canonical logical representation using logic programming, e.g., Prolog. During the analysis of 25 different approaches to formalize legal knowledge into a LES "There can be little doubt, then, that the successful construction of expert systems in law will be of profound theoretical and practical importance to all those whose concern is the law." (Susskind, 1987, p. 194). He emphasized the importance of LESs, not only to computer scientists but especially

to those working in the field of law. One drawback he identified at that time was the lack of systems that sufficiently support analogies and indeterminacy within statutes.

Based on the requirement to support legal reasoning on analogies, the work of Rissland and Skalak (1989) set a milestone. They addressed the challenge to automate statutory interpretation, which to a large degree consists of interpreting "under-defined" terms. They combine the groundbreaking results from Case-based Reasoning (CBR) (see Ashley and Rissland 1988; Ashley 1991) with rule-based reasoning in a system called "CABARET" (CAse-BAsed REasoning Tool). The main idea is to use CBR, where formal expressions of conditions cannot be applied due to imprecise words or phrases. It becomes clear that rule-based reasoning, and CBR and LES are complementary methods with pros and cons.

At the same time when the LES were most highly in demand, there were several approaches in Germany, summarized by Jandach (1993). He not only constructively differentiated between the approaches that have been made, e.g., HYPO, CABARET, TAXMAN, etc., but also provided a methodological framework for the construction of LES. Jandach summarized different main (software) components required to comply with the modern understanding of a system that represents formally modeled legal knowledge.

Meanwhile the scientific field on the rule-based inference for legal reasoning has evolved, and rule-based systems in industrial applications are omnipresent. Two main software tools were established over the last years: i) the Oracle Policy Automation (OPA) from the commercial vendor Oracle Inc. (2017), and ii) Drools from the open-source community redhat Inc. (2017). Especially the OPA is used to formalize large decision structures for governments and public institutions. In addition, some attempts by smaller companies have been made to provide a formalization of decision structures focusing on legal reasoning, e.g., Neota Logic[5].

In addition to approaches to improve the performance of rule-based reasoning engines, successes were achieved in improving legal reasoning on a formal representation of legal knowledge. So-called legal ontologies were used to more adequately reflect legal knowledge (see Wyner 2008; Casellas 2011; Sartor et al. 2011a). Improvements in knowledge representation led to the usage of ontologies, not only to structure the domain knowledge within legal systems, i.e. taxonomies, but also to provide means to describe the types, i.e. concepts, with their attributes and relationships. Expressing logical constraints and relationships between those knowledge objects can be done with description logics, e.g., Web Ontology Language (OWL). OWL allows define constraints and axioms in ontologies. Many prior attempts used the W3C standard for modeling ontologies, are based on the Resource Description Framework (RDF) and OWL. OWL lacks the possibility to formalize arithmetical or complex logic operations. Since OWL is an description logic, it was not designed to be used for arithmetical expressions or to express higher order predicates.

2.3.3. User-oriented Decision and Reasoning Systems

Enabling users to understand, analyze, and model relevant use cases, e.g., user stories, has become an important paradigm in the domains of software engineering. Therefore, the focus lies on providing modeling languages, i.e., notations, that are expressive enough to capture all

[5]https://www.neotalogic.com/, accessed on September 3, 2018

relevant issues of a particular domain, while remaining simple enough to be used by users. Well-known examples of those user-oriented modeling languages are Unified Modeling Language (UML) (Object Management Group, 2011b), Business Process Modeling Notation (BPMN) (Object Management Group, 2011a), Case Management Modeling Notation (CMMN) (Object Management Group, 2014), or Decision Modeling Notation (DMN) (Object Management Group, 2015).

UML has become a de facto standard in modeling and representing (software) systems. Various static diagrams, such as class diagrams and dynamic diagrams, e.g., data flow diagrams, are available to capture the different semantics of a software system (Object Management Group, 2011b). There have been several attempts to enrich UML with formalized and executable rules (Leon, 2001). However, UML is was never designed for modeling complex processes and work flows. Consequently, BPMN was developed. It focuses is on business modeling and specification (Object Management Group, 2011a). Many efforts have been made to adapt the BPMN to enable automated compliance checks regarding pre-defined executable rules (Sadiq and Governatori, 2015). The most recent standards provided by the Object Management Group (OMG) have a more specific focus on supporting modeling decision structures in work flows, e.g. business processes or adaptive cases. Both CMMN and the DMN, provide users with functionalities to specify complex decision structures, such as decision tables (Object Management Group, 2015, Clause 8). DMN furthermore specifies a Friendly Enough Expression Language (FEEL) (Object Management Group, 2015, Clause 9-10), which is a side-effect-free, i.e. functional, expression language based that implements a simple data model (numbers, dates, strings, lists, and contexts) and a ternary logic. The intended purpose of the simple syntax is to address a wide audience, i.e., users (Object Management Group, 2015, pp. 85).

This brief sketch of the development shows how important the user orientation and empowerment has been for the success of modeling notations. It also shows, that the provision of executable semantics has always been an important part of the standardizations efforts. Within the last years, the efforts to formalizing work flows and decision structures have intensified. This heavily increases the transparency of business processes and adaptive cases, therefore allowing optimizing and leveraging of efficiency and effectiveness.

In this thesis however, the integration for end-users plays a minor role with regard to visualizations and modeling notations. The main focus is to explore the potentials that arise during the support of interpretation on semantically annotated legal texts, which is of fundamental significance for the subsequent representation in one of the mentioned notations. As UML is an accepted standard for modeling of complex systems and can also be used to model ontological aspects, it will be applied throughout the thesis to visualize and structure the models used for computational reasoning.

2.4. Summary

Based on this overview, the relevancy of decision support systems is once again underpinned and the combination of state-of-the-art legal text analytics software and a legal reasoning framework is established. This thesis is as an additional contribution to narrow the gap between the textual

representation of decision structures on the one hand, and formal representation to allow legal reasoning on the other hand. The overview identifies the research potential at the intersection of legal informatics and software engineering, focusing on the legal domain in Germany. This potential can be identified in three complementary fields:

- **Text analytics and classification of legal norms:** From the set of available technologies, frameworks, and software architectures this thesis investigates which one should be used for legal text analytics. Thereby, the focus lies on supporting the various use cases that are relevant within the field of legal informatics, especially the extraction of information from various legal documents and the classification of legal norms. A main concern is to foster the reuse of software components and trained analytics models.

  This thesis investigates different platforms and concepts of how modern applications for text analytics are designed and concludes by recommending the use of an open-source framework, namely Apache UIMA. However, to fulfill the particularities of the German legal domain, the design decision, with regard to the data model and type system needs to be explicated and addressed.

- **Computational models of legal reasoning on German statutes:** The creation of computational decision structures that capture the semantics, as they are described in German statutes, can partially be done using ontologies. The standards in modeling ontologies lack several functionalities, such as arithmetical reasoning, which are identified and addressed by this thesis. In addition, it is important that the decision structure reflects — to the largest degree possible — the content as provided in the statutory text.

  This thesis uses the well-studied base line of ontologies on structure and reason on legal knowledge. However, Web Ontology Language (OWL) does not suffice for the representation of the semantics of legal norms in a computational manner. The usage of description logic can be considered as an acceptable step for knowledge representation, but is not adequate for more comprehensive reasoning structures.

- **Synthesis of text analytics and computational models:** The integration of text analytics components and a full-stack implementation of a system that allows reasoning on model-based decision structures offers possibilities to link parts of the computational model directly with its interpreted source in the legal document. This synthesis contributes to the well-studied problem of isomorphism. The usage of automatically and manually created annotations that explicate the semantics of statutes allow their linkage to corresponding parts in computable decision structures.

  This thesis describes the continuous spectrum of structuring the unstructured content in German statutes by automatically adding additional semantics, and by explicating how this semantics lead to computational decision structures that are linked to the textual parts from which they emerged. These links increase the transparency of the decision structure and allow for plausibility checks with regard to the semantic equivalence of the text and the computational model.

The next chapter will introduce the methods and components for legal text analytics in detail and will discuss their role and particularities for the analysis of German legal documents.

CHAPTER 3

Semantic Analysis and Annotation of Legal Documents

This chapter describes semantic analysis and annotation of legal documents using software. A reference process is proposed for this purpose which has successfully been validated in an industry project. The reference process describes the interaction between software engineers, legal experts, and software services in order to semantically annotate legal documents in an interdisciplinary setting (see Section 3.1).

The semantic analysis of legal documents is a complex and challenging task. This task requires interdisciplinary capabilities throughout the entire process. This holds true, although the technological possibilities have dramatically increased over the last decades. It is still important to not only implement and conceptualize what can be done using modern tools and software components, but to investigate the human aspect concurrently. This includes especially the people with their skills. In the field of legal informatics in particular, the semantic analysis is primarily done for humans who later on consume the information acquired by the system.

Based on these considerations for interdisciplinary legal data science, the role of semantic entities and annotation types are introduced. This can be regarded as a theoretical framework for the analysis of legal documents since it elaborates on entity types that can be extracted from legal documents. It differentiates between basic, named, and legal entities (see Section 3.2). The understanding of semantic entities within the legal domain is highly relevant since it influences subsequent implementations and requirements for a software framework that enables the analysis of text regarding these properties.

A technical introduction of strategies to annotate legal documents is provided in Section 3.3. It distinguishes between two basic strategies, namely the manual and the automated process of annotating texts. The section also elaborates on the technical implementation of two different annotation formats: stand-off and in-line annotations.

The software support for semantic analysis is comprehensively discussed in Section 3.4. It introduces different software services from computational linguistics and investigates their roles for the analysis of legal documents. It also describes the functionality in terms of different components and illustrates their applicability, but also their limitations.

Finally, Section 3.5 provides a detailed overview of existing software frameworks that allow the reuse of components for processing legal documents but also to apply machine learning techniques for the analysis of textual documents. The extensive evaluation shows that the two frameworks Apache UIMA and Apache Spark are well-suited to be re-used.

3.1. Process Model for Software-supported Semantic Analysis

Investigating different technologies that are (freely, e.g., open-source) available and can potentially be used to perform a software-supported semantic analysis, a vast number of software libraries can be found. But it is also obvious, and especially well-studied the intersection between law and IT well-studied (cf. Bench-Capon et al. (2012); Ashley (2017)), that the challenge is not only to implement, parameterize, or adapting the right software-component, but also to integrate the people with proper skills and capabilities. Within our research we have investigated the process of software-supported semantic analysis of legal documents and have formalized it based on the Rationale Unified Process (RUP) by Kruchten (2004), which allows for iterations during the proceedings and differentiates between several elements, such as:

Activities: An activity summarizes a unit of work that must be performed. The outcome results in the creation or update of artifacts, e.g., documents, datasets, or models, pattern definitions, training data, test data.

Roles: Individuals or groups performing activities of the process. In addition, roles are responsible for the artifacts that are the outcome of their activities, e.g., the legal data scientist.

Artifacts: The input and output of activities are called artifacts. They are created, modified, and used by the roles during the procedure and are either the final product, parts of it, or intermediate results, e.g., documents, models, pattern definitions.

Services, and Tool support: The software-support heavily relies on the tools and services that are used during the different activities. Operating on a given input, these create outputs that either become the input for other tools and services or for the humans involved within their responsibilities (roles) along the process.

Based on the RUP we have developed an interdisciplinary process fostering the semantic analysis of legal documents. The process has been refined during a case study (cf. Section 6.2). The process does not only take different stakeholders into account, but also the support that technology, e.g., software tools and services, are integrated and can potentially contribute. This deepens the understanding of how humans and technology can interact to solve a particular data-intensive problem in the domain of software-supported semantic analysis of legal documents. The required skills in humans and the capabilities that need to be provided by software

vary throughout the operation, since the activities that need to be performed and the artifacts differ along the process.

3.1.1. Reference Process

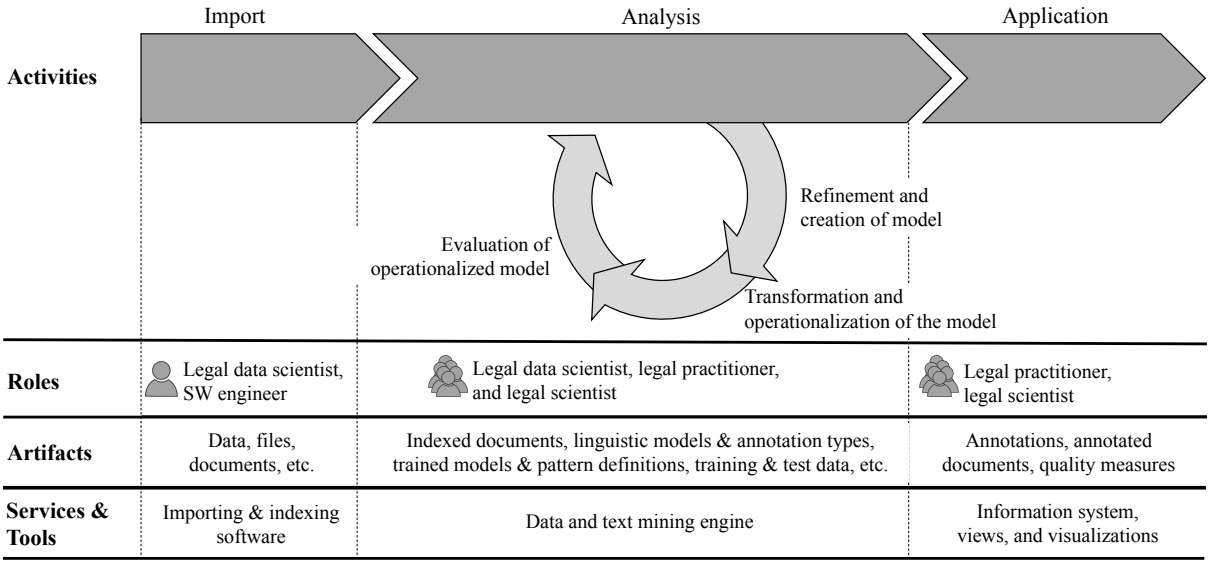


Figure 3.1.: Reference process for software-supported semantic analysis of legal documents based on Waltl et al. (2017a).

Figure 3.1 shows a visualization of the process. The illustration is structured into four different rows, namely "Activities", "Roles', "Artifacts", and "Services & Tools". The four rows are divided into three subsequent main activity columns, namely "Import", "Analysis", and "Application".

3.1.2. Activities

Activities represent the different actions performed in order to achieve a certain objective, i.e., semantic analysis of legal documents. These actions are not necessarily performed sequentially, but can be performed in iterative cycles and in parallel.

Import: The whole process starts with an import phase, in the course of which the documents stored at a specific location, e.g., URL, hard-disk drive, are retrieved. Besides the physical position of a document, the logical format, i.e., data-format, of the document needs to be read and normalized. The extraction of information from documents of various formats is still challenging. Although more data is becoming digitally available, not every document is necessarily machine-readable.

Analysis: Once the data is stored within the system and efficiently accessible, the analysis phase starts. Within this phase, the actual text mining and natural language processing operations are performed. In addition, the analysis phase has to be divided into sub-phases

that are performed as an iterative cycle. The iteration subsequently refines the results of the text mining phase and are required to ensure the quality demands of the text analytics process. This cycle needs to be performed at least once and consists of three different stages:

1. **Creation and refinement of the model:** First, the objective of the semantic analysis needs to be clear and well-defined. As discussed in Section 2, many different notions of semantic analysis exist. Therefore, it is necessary to create a model that fulfills the requirements and the overall objective, such as classification of text, classification of norms, extraction of named entities or extraction of references and dependencies, etc. (cf. Section 2.1).
2. **Transformation and operationalization of model:** Once a model has been created, it needs to be transformed into a representation, such that it is accessible by text analytics software components. This can either be a formal pattern definition, such as annotators for Pipes&Filters architectures (see Section 3.5.2), or complex models that can serve as input for (supervised) machine learning components (see Section 3.5.5). This operationalization is an important but challenging step. It is analogous to a translation process where abstract model descriptions need to be transformed into representations that can be read and understood by algorithms.
3. **Evaluation and quality assurance:** Finally, when the model has been loaded by the software and has been applied to concrete textual information, the results need to be inspected. The results can either be inspected manually (by humans), or the results are automatically compared to an existing gold standard. This inspection leads to quality measures, e.g., accuracy, precision, recall, etc., and may contribute to the decision whether an additional iteration needs to be performed. This decision must not necessarily be decided on fixed quality thresholds, but rather depends on different factors and the intended use case.

If the cyclic process has stopped the analysis phase comes to its end. The result of the cycle is a set of indexed documents, annotation types, and annotations, which reflect the extracted information (cf. Section 3.2 in combination with Section 3.3).

Application: Once the desired information is extracted and the annotations are persistently stored so that they can be accessed, the various use cases can be applied within legal information systems or search databases. The application in legal information systems includes using algorithms to apply artificial intelligence, e.g., legal reasoning (cf. Section 4). In doing so, this additional semantic information can improve the analysis and interpretation of legislative texts.

3.1.3. Roles

Throughout the reference process, different roles are required. These cover the full-stack implementation: from technological adaptations, linguistic modeling, and operationalization of the model to designing the end-user application.

Software engineer: The import phase mostly requires this for the adaption and development of new software components to retrieve the information from the documents. Whenever the documents at hand are already in a machine-readable format, such as XML or Hypertext Markup Language (HTML), the integration is much easier as the integration of highly unstructured documents or of documents with internal structures not primarily designed to represent textual information, e.g., images or pictures. Within these documents, the textual layer must be extracted, e.g., via Optical Character Recognition (OCR), before it can be imported and indexed. OCR is known to be non-trivial and prone to errors (cf. Mori et al., 1999). However, if a new set of documents is going to be imported, adaptations in the import procedures have to be implemented.

Legal data scientist: Due to the peculiarities of data science for the legal domain, so-called legal data scientists, are important during the semantic analysis. The are experts in the domains of data science and analytics. In addition, they are aware of the structure and the language of legal documents and able to combine these two expertises. This is an essential profile, since it ensures the operationalization, i.e., practical implementation, of the model.

Legal scientist: In order to derive a proper and useful model applicable for legal texts, it is necessary to deeply understand what is contained in the text and what the potential entities are that can be extracted. These entities cannot be defined by the software engineer, but require expertise in the particular legal field at hand. Therefore, the legal scientist serves as domain expert.

Legal practitioner: Just as the legal scientist, the legal practitioner is a domain expert and potential user of the application. He ensures the practicability of the implemented solution and states additional requirements that need to be met to successfully analyze the documents. In general, legal practitioners are more interested in the quality of the results, the performance of the system, and the usability of the application, rather than in the legal theory.

3.1.4. Artifacts

The process is performed on a set of digital assets, which are either the input or the output of an activity.

Data, files, and documents: The process can be considered very document-centric. It starts by importing documents and subsequently applies various operations on them. These operations are required to identify, extract, and attach additional information, which allows to recognize more complex semantics behind a particular word, phrase, sentence, section or document.

Indexed documents: Documents need to be normalized, so that they can be accessed efficiently. Once they have been brought into a machine-readable, form they are stored in an Elasticsearch (ES) database, that allows for an efficient access.

Linguistic models and annotation types: The desired information that should automatically be extracted from the documents needs to be formalized according to linguistic models and

annotation types. This heavily depends on the use case to be realized using the software. Use cases can reach from very-low level extraction of the nouns and keywords of a document, to identifying norm categories in statutory texts, such as legal definitions, obligations, prohibitions (see Section 3.2).

Training data & test data: The procedures used during the semantic analysis mostly need training and test data sets. These are needed to either train supervised machine learning procedures (see Section 3.5.5), or to test and measure the quality of the applied procedures.

Pattern definitions & trained models: During the analysis phase, models are created, applied, and constantly refined. These models are used to create extract the desired information from the documents. The concrete technical implementations of these models vary. Whereas rule-based text analytics models consist of a set of potentially nested rules, more complex models for machine learning consist of state representations of trained algorithms. These representations are no longer readable or maintainable by humans.

3.1.5. Services & Tools

As already mentioned, the process can be considered document-centric. One of the key elements are the software tools and services that assist and automatically perform the semantic analysis on these documents.

Importing & indexing software: The software requires a powerful and extensible import component. The generic platform for legal data analytics should be able to cope with documents from various sources and data types. This functionality is implemented using a generic import structure. Thereby, the different documents are normalized into one common format that is flexible enough to fully reflect the different document content structures, such as meta-data, text, footnotes, chapters, and sub-chapters. Besides this flexibility, it is necessary to have one common interface through which the different components of text analytics can be applied. This fosters the modularity of the system and the reusability of the components.

Data and text mining engine: One of the most central components of the process is the data and text mining engine. It logically groups the different software components required for text analytics. It does not only consist of the various components, but also has a flexible and generic architecture that allows to efficiently process the documents and create annotation based on the textual data contained therein. The engine takes into account textual data from the documents, but also structural data from the texts and meta-data that is attached to the documents, such as document type, creation date, author. A detailed description of the engine can be found in Section 3.5.

Database: During the process, the data that needs to be stored so that it can be efficiently accessed. In our implementation, all data is stored within an Elasticsearch database. Elasticsearch handles large amounts of textual data very efficiently and allows for fast bulk operations on the data. In addition, it supports the generic data model that underlies the process and is part of the technical implementation (see Section 3.5).

Information System: To ensure the accessibility from the results of the semantic analysis process, it is necessary to embed it into an information system. Basically, it would be possible to query the database directly, but this would be an inefficient and inconvenient way to retrieve the results and to benefit from software-supported semantic analysis of legal documents. The information system provides an intuitive front-end that can be used by users, e.g., legal practitioners, to consume the information. In addition, it allows the provision of alternative representations of the textual data, and it supports navigation, exploration and search processes in large document corpora.

3.2. Annotations, Annotation Types, and Semantic Entities

Obviously, the quality of the support during the aforementioned process depends on the different artifacts that are created, manipulated, or analyzed in the course of the activities. In addition, the overall objective, for which the semantic analysis is performed, has a significant impact on the process. Different objectives require different artifacts.

The analysis of German statutory texts, for example, requires the availability of these documents in a digital format. Trivially, documents that are not present, or not in a valid format, cannot be investigated using information extraction methods. The same argument holds for information that is not contained within the documents. This circumstance is often overseen by critiques; but enthusiasts also forget about this obstacle fact from time to time. How the presence or absence of information is interpreted, is a different question.

As discussed in Section 3.1.2, every process requires a linguistic model, that formalizes and codifies the objectives of the analysis to be performed. The concrete implementation of this model can have different notions, and it depends on the underlying system with its capabilities and architecture. However, it is necessary to create a linguistic model that specifies the semantic entities that should be contained within the documents. These semantic entities reflect the model and are not supposed to be a stable set, but they are rather flexible and need to be adapted.

The main factors that influence the adoption of models lie in the very nature of the models themselves, which can be derived from Stachowiak (1973):

1. **Domain:** Different domains require different semantic objects. On a semantic level, analyzing statutory texts differs from the analysis of contracts or emails. There might are be commonalities, but the entities that are contained within cannot be reused without reflection and adaption.
   Examples for domains could be: statutory texts, judgments, emails, and contracts.

2. **Objective:** Different objectives require different semantic objects. Every semantic analysis process is performed for a particular reason. If the reason changes, the linguistic model needs to be adapted. In the case studies of our evaluation, we will show (see Section 6), that the semantic analysis to support the editorial staff of a company differs from providing interpretation support for legal experts.
   Examples for objectives could be: risk analysis (in contracts), support of editorial staff, extraction of definitions in judgments (e.g., Walter 2010).

3. **Time:** As documents, language, structure, or the way how textual information is expressed in general changes over time, e.g., by introducing new words or concepts, the model needs to be adapted.
   An example could be the analysis of time series data in which the language has changed over time.

Based on these three influence factors, we can conclude that a model always depends on the context of its usage. The reuse of a model does not pose a problem per se, but requires thorough assessment of validity and applicability.

Several attempts have been made to provide an abstract and generally acceptable model of entities within legal texts (see Hoekstra et al. 2007; Casellas 2011; Sartor et al. 2011a). However, these models have hardly been adopted and they can only partially be re-used. This lack of reuse can be traced back to the above mentioned drawbacks and concerns. The lack of a general re-use for linguistic models lies in the very nature of models themselves and was already observed by Stachowiak (1973) decades ago. He showed that models are always

1. mapping the variety of the real world,
2. reducing the complexity of the real world, and
3. created to serve a pragmatic purpose.

Based on these considerations, the creation of models is always preliminary to a certain degree. However, models are the base line for semantic analysis and we will propose a technological framework and architecture providing end-users with the full flexibility to represent linguistic models with their particularities.

3.2.1. Annotations and Annotation Types

The recognition and extraction of semantic types, e.g., obligations, prohibitions, definitions, etc., in legal documents is technically done using annotations: a particular region of the document, e.g., text, images, etc., is addressed and additional information is attached (see Section 3.3.4). For larger documents and advanced information extraction processes, the amount of annotations attached to one document can grow very quickly. Therefore, an efficient way of handling these annotations needs to be implemented. For this management of documents, the UIMA (see Ferrucci et al. 2009) seems to emerge as a standard in the industry[1] and sciences (see Grabmair et al. 2015).

However, the UIMA only handles the internal (technical) representation but does not create annotations by itself nor does it create or recommend annotation types. This input is domain dependent and has to be provided by a domain expert, such as a legal scientist, a legal practitioners, a (legal) data scientist, or a linguist (see Section 3.1).

Within our research, we use two terms that need to be differentiated, namely:

[1] https://www.ibm.com/developerworks/data/downloads/uima/index.html, accessed on September 3, 2018

Definition: Annotation
An annotation is an object which attaches additional information, i.e., meta data, to a particular region of the subject of analysis by containing the assigned information in a structured format and unambiguously referring to the intended region within the subject of analysis.

Definition: Annotation Type
An annotation type is an object that is associated with an annotation and defines its semantic role on an abstract level.

The excerpt below shows an annotation of the type "Definition" and the annotated text which is the first sentence from Section 90 of the German civil code. This is an abstract example of an annotation and the corresponding annotation type.

Example
Annotation Type: Definition
Annotated Text: Only corporeal objects are things as defined by law.
Annotated Region: §90 German Civil Code, Sentence 1

The concept behind annotations and annotation types is rather simple and can be explained using another example as shown in Figure 3.2. In the textual representation, displayed in the middle section, each part of the text to which an annotation exists is highlighted. In a subsequent processing step, algorithms have analyzed the text. An annotation for sentence that is a legal definition was created. These annotations belong to the annotation type "Legal Definition". The list of determined annotation types can be seen on the left side of the screenshot. The front-end differentiates between "Linguistic Entities" (see Section 3.2.2), "Named Entities" (see Section 3.2.3), and "Legal Entities" (see Section 3.2.4).

To handle complex named entities and legal entities, annotations are subsequently combined and aggregated. Starting with the detection of basic and linguistic entities, e.g., extraction of tokens and sentences, the system can detect noun phrases and names of, for example of persons and organizations. Based on that, rather complex legal entities, such as legal definitions, rights, obligations or permissions, can be determined.

It should be explicitly noted that multiple annotations can refer to the same lexical unit. For example, "house" is a token, a word, and a noun. which can lead to three different annotations. Consequently, annotations do not have strict order, e.g., hierarchically or sequentially, but can arbitrarily overlap. The next section will elaborate on different basic and legal entities that can be found in legal documents.

3.2.2. Basic and Linguistic Entities

Basic and linguistic entities are formed by various kinds of rather technical and low-level annotation types. Typically, these are intermediate results and contribute to a semantically more

Figure 3.2.: View of an automatically annotated German law (left: annotation type selection using check boxes; middle: annotated and highlighted text; right: labels visualizing the annotations separately).

advanced entity. The raw outcome of basic software components used during the annotation process belong to this type.

Representatives are:

- Punctuation
- Special characters (e.g., §, $, &)
- Tokens
- Words
- Significant words and phrases (e.g., stop words, auxiliary sentences)
- Sentences
- Part-of-speech tags (e.g., nouns, verbs, adjectives, adverbs)
- Paragraph
- Document

Basic and linguistic entities are very important. They can be considered the foundation for more complex annotations. The name "basic entities" should not be misleading. Although their extraction and annotation might seem to be easy (or very low level), achieving the highest performance is not trivial, considering that heterogeneous datasets, variations in language or spelling, erroneous text in digital documents, etc. have a negative effect on the recognition rate.

3.2.3. Named Entities

The recognition of information units with a particular names is an essential part in the domain of information retrieval. There is no commonly accepted notion of the "named entities" (see Nadeau and Sekine 2007). During the last years, the term "named entity" has broadly been applied in information retrieval and natural language processing.

Representatives are:

- Persons
- Organizations
- Geographical names and locations
- Expressions of time
- Quantities
- Monetary values
- Percentages
- References and citations
- Docket numbers

The recognition of named entities is extensively discussed in Section 3.4.1.4.

3.2.4. Legal Entities

Semantically analyzing legal documents requires the specification of legal entity types, or, in short, legal entities. These form the third group of annotation types, which is particularly tailored to the legal domain, whereas the other entity types are, more or less, domain-agnostic.

Different attempts exist to generally and ultimately define legal entities. These models are either very abstract, or only partially applicable (see Hoekstra et al. 2007). A more general discussion can be found at the beginning of this section.

However, depending on the data and the objective of the semantic analysis, the representatives of legal entities as defined by Hoekstra et al. (2007) are:

- **Norms:** Describe on an abstract level generic situations and define a specific state or action. Their description is qualified by a deontic term.
- **Permissions:** Are a special type of a norm, as they allow allow something, but do not prohibit anything.
- **Prohibitions:** Are a special type of a norm, as they prohibit certain qualified actions or situations.
- **Obligations:** Are a special type of a norm, as they make certain qualified actions or situations mandatory.

- **Rights:** Are norms that assign and grant certain qualified actions or situations.

A recent and comprehensive taxonomy, plus a detailed description of entities, i.e., "semantic types", for the analysis of US case law was provided by Walker et al. (2017):

- **Citation (sentence or clause):** A citation sentence is a sentence whose primary function is to reference legal authorities or other materials.
- **Legal-rule (sentence or clause):** A legal-rule sentence is a sentence that primarily states one or more legal rules, without stating whether the conditions of the rule(s) are satisfied in the case being decided.
- **Legal-policy (sentence or clause):** A legal-policy sentence is a sentence that primarily states one or more legal policies, principles or objectives.
- **Policy-based-reasoning (sentence or clause):** A policy-based-reasoning sentence is a sentence that primarily applies legal policies to decide legal issues.
- **Ruling or holding (sentence or clause):** A sentence that states a ruling or holding is a sentence that primarily states, "as a matter of law", whether some particular legal rule is satisfied in the case.
- **Rule-based-reasoning (sentence or clause):** A sentence that states rule-based-reasoning is a sentence that reasons from a foundation of legal rules and facts to a ruling or holding as a matter of law (i.e., a conclusion of a law).
- **Evidence (sentence or clause):** An evidence sentence is a sentence that primarily states the content of the testimony of a witness, states the content of documents introduced into evidence, or describes other evidence.
- **Finding-of-fact (sentence or clause):** A finding-of-fact sentence (an evidence-based-finding sentence, or simply, a finding sentence) is a sentence that primarily states an authoritative finding, conclusion or determination of the trier of fact.
- **Evidence-based-reasoning (sentence or clause):** An evidence-based-reasoning sentence is a sentence that primarily reports the trier of fact's reasoning in making the findings of fact.
- **Procedural-fact (sentence or clause):** A procedural-fact sentence is a sentence that primarily states one or more procedural facts about the specific case, such as what motions were led or the disposition of the case at the trial level.

It becomes evident that different research groups will end up with different findings and a different set of legal entities, depending on their objective and data.

This work aims to support to provide a software architecture that allows for the integration of all the different linguistics models. The applicability and the success of this approach have already been shown by Grabmair et al. (2015) through the introduction of LUIMA, an adapted version of UIMA, to support the creation and maintenance of annotations within the legal domain. Whereas Grabmair et al. focused on the improvement of search results ranking using annotations, they did not publish an in-depth study of the software architecture and components

that are required for the extraction of legal entities and the formalization in computational decision structures, which are linked to the text source from which they emerged. This gap is addressed by the work at hand.

3.3. Annotating Legal Documents

Performing semantic analysis of textual documents has many facets that differ in the methodology, technology, and objective. In Section 2.1, we have investigated the different existing notions of semantic analysis. Based on that, we can draw the conclusion, that different tasks in the spectrum of semantic analysis have different inputs and produce different outputs. In general, the different analytical tasks have different requirements for the text to be processed and need additional resources, such as (pre-trained) models, dictionaries, pattern descriptions, knowledge bases. During the processing step, the text is analyzed and additional information is attached.

In information systems, this additional and attached information has to be technically represented in a technical way (cf. Section 3.4). For this purpose, the concept of annotations is well-studied and commonly accepted (Wilcock, 2009). This does not only hold for computational linguistics, but for the semantic analysis of documents in general. Annotations include information about the exact position of the original document to which they refers. Depending on the technical implementation, different solutions exist to store the exact position of an annotation (cf. Section 3.3.4). This basic information is required to identify the part of the text that is enriched with meta-information. The additional value of the annotation lies in the supplemental information that is attached to a region of the text. An annotation can for example hold unstructured information, e.g., a comment, or structured information about a given type that the covered text belongs to, e.g., date or money value, reference.

Basically, two different ways of creating, updating, and deleting annotations exist:

1. **Manually added annotations:** Texts that are read and interpreted by humans can be annotated to include an additional piece of information in the text. In this case, the reader creates the annotation.
2. **Automatically added annotations:** Algorithms can be programmed and trained to create annotations to given input text (cf. Section 2.1). In this case, the algorithm creates the annotation.

Handling different types of annotations while supporting their creation, deletion and updating is a fundamental prerequisite for the semantic analysis of legal documents.

3.3.1. Manually Annotating Legal Documents

During the interpretation and reading of a particular legal document, humans can create associations that are induced or caused by a concrete piece of text. These interpretations might be valuable to other readers or to the same reader at a different point in time. In the analogous world, the reader would take a pen or a marking pencil and write the annotation on the sheet of paper. This straight-forward process has to be represented by the system in order to enable manual annotation of legal documents. During the reading of a document, users have the possibility to mark a particular piece of text and subsequently attach additional information (represented by annotations).

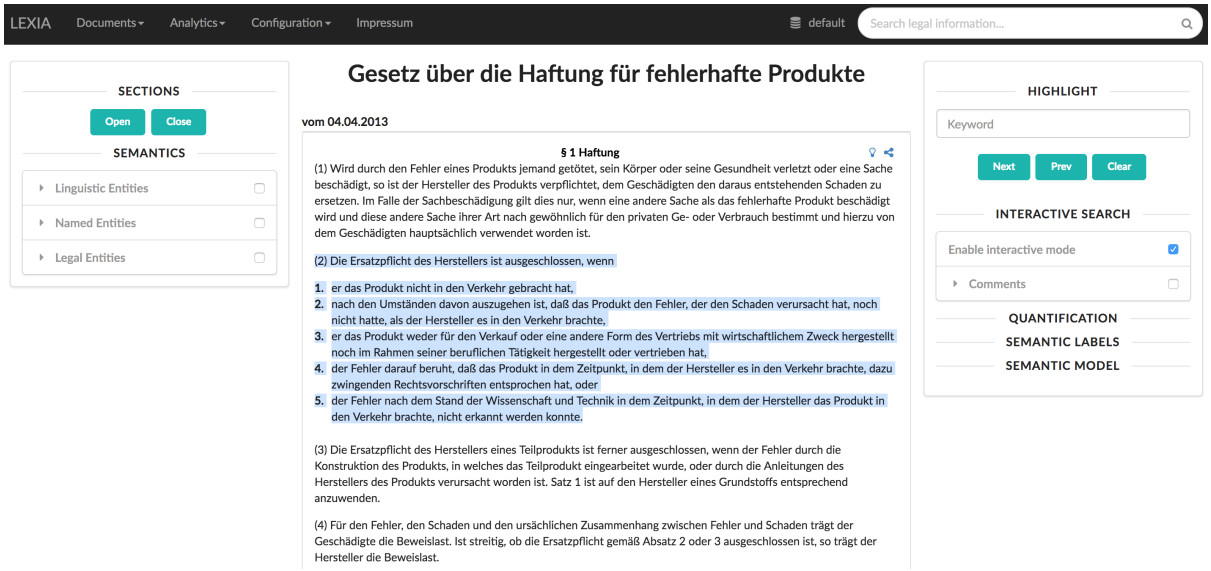


Figure 3.3.: Manually highlighted text to be annotated.

Figure 3.3 shows a screenshot of the implemented system and how manual annotations are created. By enabling the "interactive mode" (see right side), the user can select the text from the middle part, to which an annotation is going to be created. The system automatically determines the proper offsets and covered text to unambiguously assign and keep the annotated region of the legal document.

Figure 3.4 shows the next step in the workflow of manual annotation. Once a text has been selected a pop-up appears which allows the user to provide additional information. The user can either create a comment as a free text information, or the user assigns an annotation type to the covered text. These types are explicit meta information that classify the covered text as a particular semantic type, e.g., reference, money value, date value, legal definition. Possible annotation types are introduced in Section 3.2.

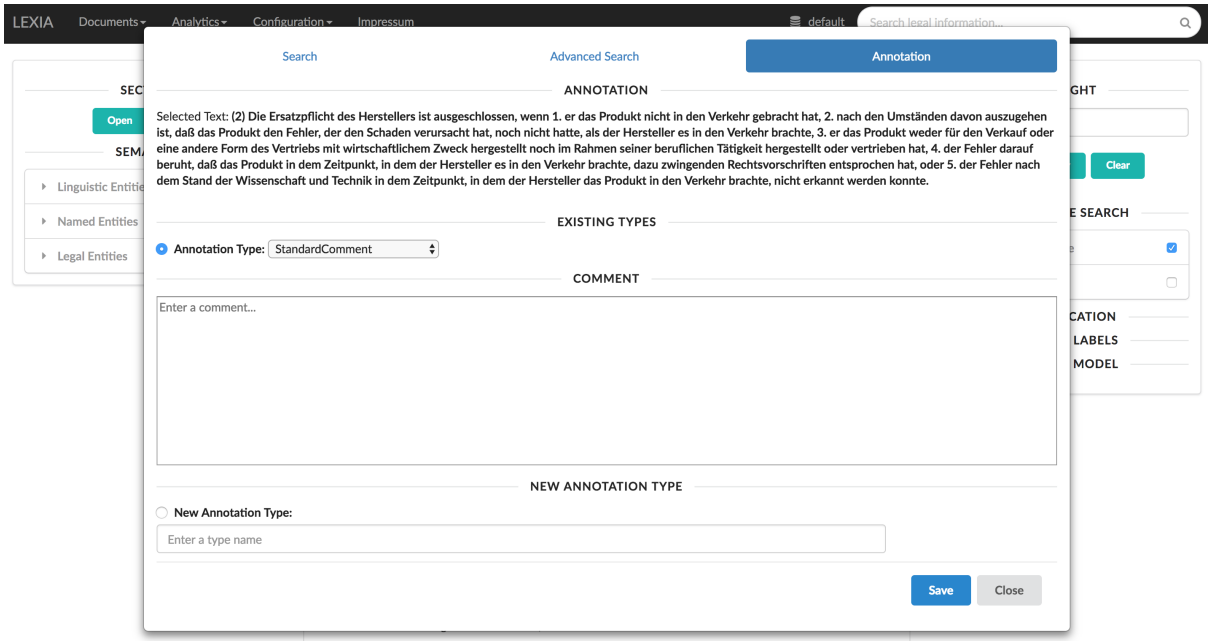


Figure 3.4.: Pop-up allowing the annotation of the selected text with freetext information or by assigning a semantic type.

3.3.2. Automatically Annotating Legal Documents

Beside the possibility to manually create annotations, the option to automatically annotate legal documents is more relevant. In Section 2.1, we introduced several technologies that can (semi-)automatically analyze textual information and recognize and extract relevant textual parts.

Although text mining components perform different tasks using distinct techniques, the underlying workflow remains the same — at least from a technical perspective: Based on given textual data and an optional pre-trained model, the text mining component analyzes the text and creates a set of annotations that directly or indirectly refer to the input data.

Different software architectures exist for this purpose, enabling the automated processing of documents and meeting the highest standards in performance, modularity, and usability. We have made an extensive study of different existing software architectures (see Section 3.5) and have chosen the open source software architecture UIMA, which is part of the Apache Software Foundation.

Figure 3.5 shows a screenshot of an annotated legal document. Therein, the textual representation of the original text is enriched with annotations, rendered as colorful boxes surrounding the textual phrase to which an annotation belongs. Different annotation types are distinguished with different colors. In our implementation the colors are determined by the name of the annotation type; the title of an annotation type is hashed and then mapped into the RGB space. This procedure assigns a color to an annotation type. The chance that two different annotations share the same color is very unlikely, since the hash function uses String.hashCode() function

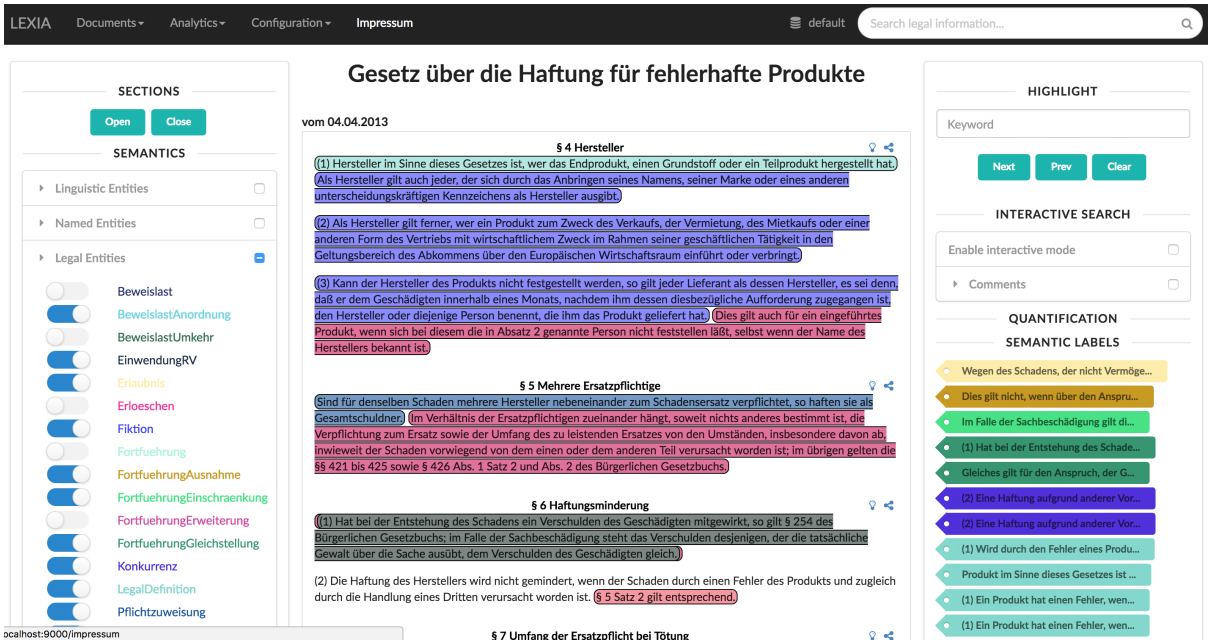


Figure 3.5.: An example of an automatically annotated German law (excerpt from the product liability act).

of JavaScript. This ensures the immediate, visually pleasing feedback of the annotated content and allows users to instantly check the annotation's validity.

3.3.3. Collaborative Maintenance of Annotations

The implemented system allows for basic features of a collaborative information system. As discussed in Section 3.3.1, users can manually add new annotations and thereby create an annotated document. However, due to its implementation as a web application, the system enables multiple different users to not only share, but also to maintain the different annotations together. The annotations inserted by another user can be accessed and manipulated, i.e., deleted.

Figure 3.6 shows the pop-up that appears if an annotation box is clicked at the front-end. Within the pop-up, different information is shown: given a specific annotation, it shows some general information about the semantic entity, i.e., annotation type, that is selected. In the mentioned Figure, this is "Fiktion". In addition, it shows the covered text. To support the exploration of the indexed documents and to determine potentially relevant dependencies, the system shows recommended annotations. These are determined using two different technologies: i) "More-Like-This" functionality of the underlying database, i.e., ES[2], and ii) Locality-sensitive hashing, as described by Leskovec et al. (2014).

[2]https://www.elastic.co, accessed on September 3, 2018

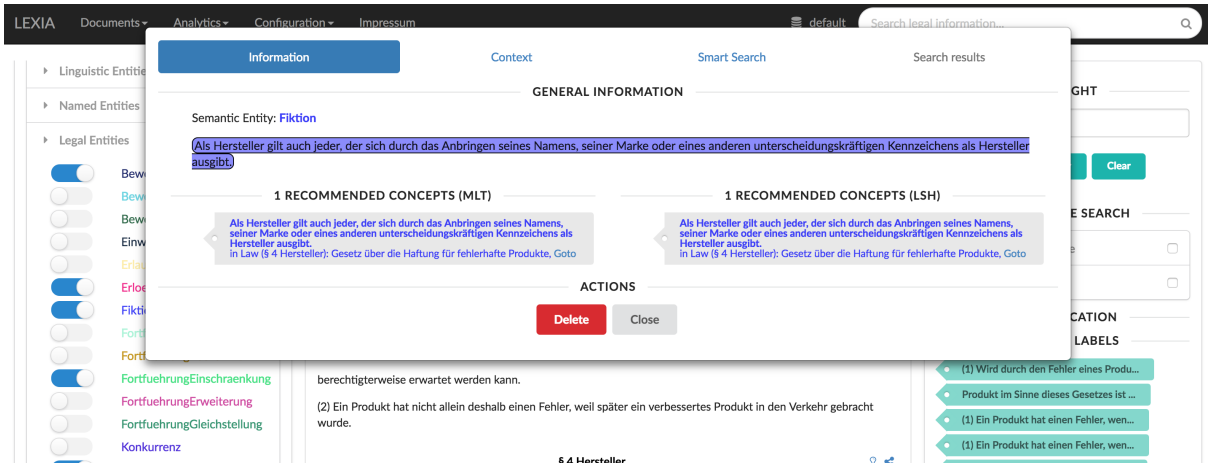


Figure 3.6.: Annotations can also be deleted (see the button in the lower part of the pop-up).

3.3.4. Annotating Legal Documents: a Technical Perspective

From at technical point of view there are two fundamentally different ways to attach this additional piece of information, as discussed by Grass (2014) and Wilcock (2009):

1. **In-line annotation:** Additional information is embedded directly into the text. This is the basic mechanisms of standard mark-up languages, such as XML or HTML.
2. **Stand-off annotation:** Additional information is a separate object attached to the original text. In this case, the annotation object contains the information required to resolve the mapping towards the original text.

Both approaches have advantages and disadvantages, which have been subject to extensive studies (e.g., Grass 2014).

Criteria	In-line	Stand-off
Adding semantic information	+++	+++
Interoperability with other formats (CSV, JSON, etc.)	++	+++
Non-manipulation of original text	++	+++
Allowance of overlapping annotations	-	+++
Self-containing (no separate file)	+++	-
Analytics of annotations	++	+++
Machine-readability of annotations	+++	++
Coping with changes within original text	+++	+

Table 3.1.: Comparison of the in-line and stand-off annotation methods.

Table 3.1 summarizes the advantages and disadvantages based on seven criteria. These mainly target the operationalization of annotations within a web application focusing on the semantic analysis of textual documents. Especially the lack of overlapping annotations hinders the usage

of in-line annotations. The tree-like structure of XML is perfectly suited to work on annotations that fully contain each other, such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<sentence>
A B C <phrase> D E F </phrase> G.
</sentence>
```

However, annotations that overlap each other, which could occur in manually added annotations, e.g., comments, but also in automatically added annotations, e.g., POS–tags, linguistic phrases, norm types, etc. lead to inconsistencies within the XML format and consequently to invalid XML. An occurrence of overlapping annotations is provided in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<sentence>
A <comment> B C <phrase> D E </comment> F </phrase> G.
</sentence>
```

During a syntax-check of XML validity[3] the listing above produces an error. XML validators will check the nestedness of the XML elements and would produce an "error on line 2 at column 52: Opening and ending tag mismatch: phrase line 0 and comment". The listing closes the phrase tag with a comment tag, which is not valid for in-line annotations. However, this is a constraint that cannot be accepted during the enrichment of textual data.

There are several strategies to cope with that issue, such as resolving the overlapping annotation by properly inserting opening and closing tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<sentence>
A <comment> B C </comment> <phrase> <comment> D E </comment> F </phrase> G.
</sentence>
```

The listing above shows valid XML, but introduces additional complexity in creating and interpreting the XML document. Within the implemented system, the annotation object is one of central objects that is either going to be created manually (cf. Section 3.3) or automatically by NLP technology (cf. Section 3.3.2). Based on these studies, we decided to build our prototypical implementations using stand-off annotations. The impact of this decisions, especially for the data model, is thoroughly discussed in Section 3.4.

3.4. A Software Architecture for Managing Annotated Legal Documents

The importance of annotations and their role during the software-supported semantic analysis was explored in Section 3.2. The focus of this investigation was set on the variety of different annotation types that can exist when it comes up to the analysis of legal documents. These either fully reflect the particularities of legal documents, such as annotations of legal definitions, prohibitions, obligations, or they are of more general or technical nature, such as tokens, sentences,

[3] https://www.w3schools.com/xml/xml_validator.asp, accessed on September 3, 2018

or references. The subsequent Section 3.3 has elaborated on the maintenance of annotations and the different ways of doing so; namely their manual and automatic creation, deletion, and manipulation of annotation, were discussed.

This section introduces a software architecture capable of handling the various operations for annotating legal documents. Therefore, the focus has explicitly been set on the provision of a generic framework, that is easily extensible (see Section 3.4.1). The framework, a fully implemented research prototype, is called LEXIA, which is the acronym for Legal Information Analysis, Exploration, and Reasoning Platform. It consists of loosely coupled software components that interact with each other via well-defined interfaces. The requirement of extensibility does not only hold for the components, but also to the data model. The system has been designed to follow a modular structure that allows for the seamless integration of new document types.

3.4.1. Software Components for Semantic Analysis

The Legal Information Analysis, Exploration, and Reasoning Platform (LEXIA) has been under development since 2014 and is primarily designed to support various forms of semantic analysis for German legal documents. We extended the functionality in different proof-of-concepts and case studies (see Section 6). The main architecture was published in two peer-reviewed article: Waltl et al. (2015) and Waltl et al. (2016).

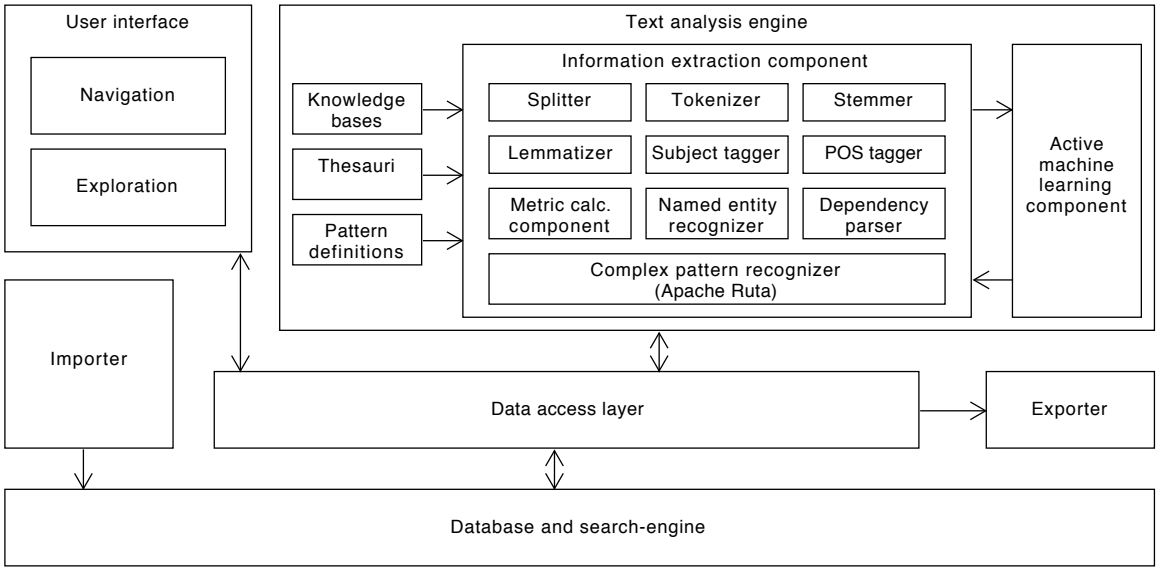


Figure 3.7.: Overview of LEXIA's system architecture with the main components (originally published in Waltl et al. 2016).

LEXIA is a web application integrating basic concepts of collaborative information systems. LEXIA's back-end is a Java-based web framework, namely the Play Framework. The Play

Framework is an established web application framework that claims to be a "lightweight, stateless, web-friendly architecture" (see Play Framework 2017). It is released under the Apache 2 License and follows a Model View Controller (MVC) design principle. The front-end is implemented using JavaScript and AngularJs.

Figure 3.7 shows the overall high-level architecture with the different components. It also shows the main information flow between the components. The details about each of the component are provided in the subsequent Sections 3.4.1.1 – 3.4.1.5.

3.4.1.1. Importer and Exporter

The importing structure maps the documents that need to be parsed and indexed, and which can be of any data type (PDF, XML, etc.), into the data model of our system. For this purpose, the importer has to be flexible enough to support various digital formats.

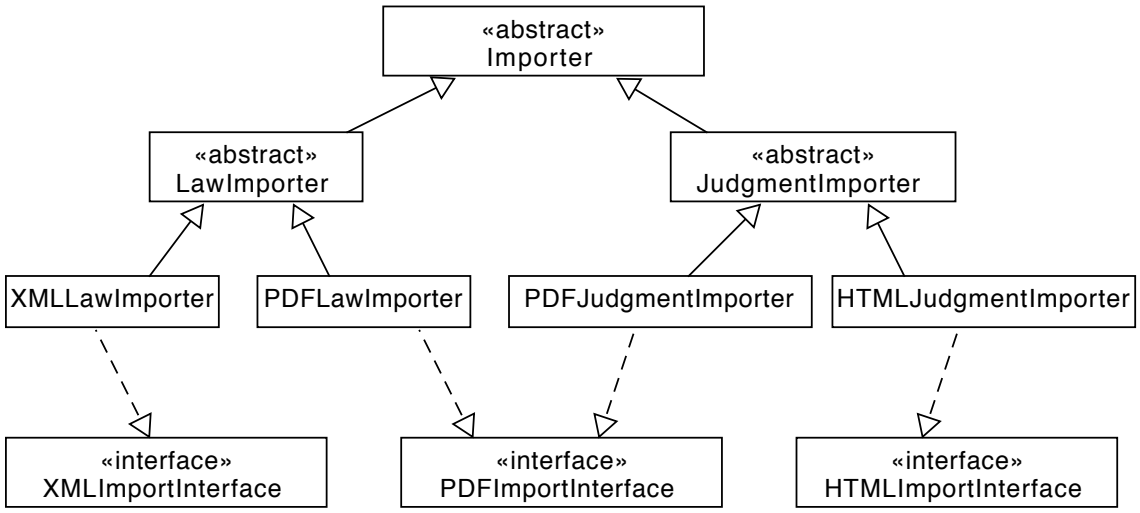


Figure 3.8.: Import architecture to flexibly support different documents types, e.g., laws, judgments, and document formats, e.g., XML, PDF, and HTML (see Waltl et al. 2016).

Thus, we propose the architecture as shown in Figure 3. The inherited classes LawImporter and JudgmentImporter differentiate between two document types we currently support in our system. Those again serve as base classes for the concrete importing mechanisms, responsible for the parsing and transformation, e.g., XMLLawImporter, PDFJudgmentImporter. Due to the inheritance, a logical separation has been achieved, so this architecture does not only allow to easily extend to new documents types, but also to new data sources and data channels (e.g., REST API, open data resources).

To standardize the different file format importers, we have provided interfaces for each file format. If, for example, a new component is required to import judgments from XML files, this new component would be derived from the JudgmentImporter class and simultaneously implement

the XMLImportInterface. This would standardize the access to this particular component and the remaining importing procedures, i.e., storing in the database, would be performed automatically.

The exporter component provides interfaces for other applications (e.g., REST API) to use and reuse the information stored. In addition, the implementation provides methods to create data dumps (CSV) that can for example be used to conduct advanced (statistical) analysis of the analyze document corpus in appropriate tools, e.g., Matlab, or R. Both components, the importer and the exporter, support bulk operations.

3.4.1.2. Data Storage

The importing structure inserts extracted data directly into the data storage, which consist of an Elasticsearch database (see Elasticsearch 2017). Elasticsearch is a distributed search engine which was developed in Java and released under the terms of the Apache License. According to international rankings it is one of the most popular search engines[4]. Elasticsearch supports multitenancy and near real-time search. It can distribute the stored data over multiple hosts, and maintains the records in so-called primary and replicated shards. Rebalancing of the records as well as routing between hosts is done automatically.

Elasticsearch is accessed via a REST API and provides a ready-to-use Java API. The data can be retrieved using real-time Hypertext Transfer Protocol (HTTP) GET requests. Elasticsearch is considered to be a NoSQL datastore.

In our implementation Elasticsearch serves as datastore and as full-text search index. Due to its native support of complex queries, it decreases the effort of integrating full-text search in the user interface. In addition, it allows to easily add convenience features, such as faceted search, and auto-suggestions. As shown in Section 3.6, it also offers an out-of-the-box recommendation component, namely More-Like-This.

3.4.1.3. Data Access Layer

All operations on the data storage, i.e., create, read, update, and delete, that are initiated in the user interface or in the text analysis engine, are handled by the data access layer. Parts of the business logic and the data model (see also Section 5.1.2) are implemented therein. The data access layer accesses the REST API of Elasticsearch and maps the documents stored within into Java objects. In addition, it also provides methods transforming Java objects into JavaScript Object Notation (JSON) objects, so they can be handled on the user interface, i.e., front-end.

Furthermore, the data access layer provides methods to access Elasticsearch-specific functionalities such as full-text search, faceted search, and More-Like-This.

The data access layer also provides an interface for bulk operations. These allow the simultaneous manipulation, i.e., insertion, creation, update, and deletion, of a set of documents, and ensure efficient handling of larger data sets by the text analysis engine.

[4] www.db-engines.com, accessed on September 3, 2018

3.4.1.4. Text Analysis Engine

The text analysis engine is considered to be the core of the implementation. It contains various software components required to perform the semantic analysis of legal documents. These components are either fully written in Java and contained in the system, or accessed via an Application Programming Interface (API). This section gives a brief overview of the different components and their basic functionalities. It provides a static view on the systems architecture. The description of the processing is provided in Section 3.5.

Knowledge Base, Thesauri, and Pattern Definitions

Although many NLP software components are self-contained, i.e., do not required additional information input from an external knowledge base, there are several components that still need to get input from external sources, e.g., dictionaries, or require explicit training, e.g., pattern definitions.

LEXIA handles different kinds of these components. On the one hand, it is possible to access open-data knowledge bases, such as Wiktionary[5] or Wikipedia[6]. On the other hand, resources that require access to information that is locally stored on the server, e.g., dictionaries, and rule definitions, can also be handled by our system. Thereby, the system allows to flexibly maintain these resources via the user interface.

The resources are shared with everyone who has access to the system. This fosters the collaborative creation and maintenance of resources. Currently, the system does not have any restrictions regarding access control management.

The systems has pattern descriptions for the following components:

- LawInboundReferencePattern
- LawOutboundReferencePattern
- LawSegmentationPattern
- LawArticleHeaderPattern
- JudgmentReferencePattern
- JudgmentSegmentationPattern
- LeitsatzPattern
- ZitatPattern

[5]https://www.wiktionary.org, accessed on September 3, 2018
[6]https://www.wikipedia.org, accessed on September 3, 2018

3.4.1.5. Information Extraction Component

The information extraction component handles various core components of NLP. These components are mostly implemented as UIMA components, so that they fit seamless into the structure and Pipes & Filters architecture (see Section 3.5.2).

The next paragraphs introduce the main components and describe their core functionalities. Most of the components have not been implemented within this research project, but rely on publicly available and open source implementation. A large collection of components can be found at the repository of DKPro[7].

Splitter

During the splitting process, the document is divided into different sections and logically connected parts. The splitter deals with document content that also contains XML mark-up. Depending on the concrete use case, the size of the different parts may vary. In most of our use cases, we split the text into sentences (see Figure 3.9), which can be tokenized afterwards. Other use cases may require a splitting into paragraphs or larger sections, e.g., chapters.

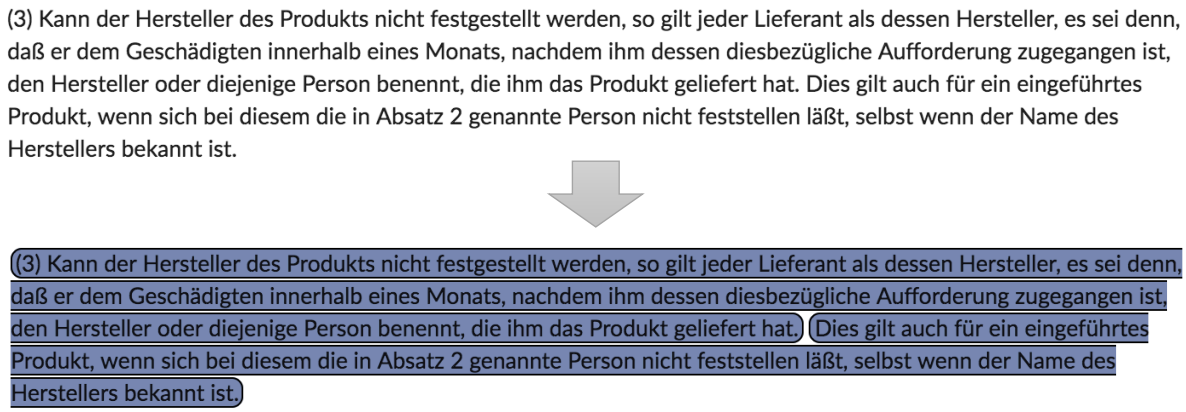


Figure 3.9.: An example of splitting article 1 of the product liability act on the sentence level.

Although this task seems to be a straight-forward operation, recent research (see Savelka and Ashley 2017) has shown that the accuracy of software in performing tokenization and sentence splitting hardly reaches 1, i.e., 100%.

There are different strategies to split the document into sentences. Our implementation supports the usage of context free grammars, e.g., regular expressions. Splitting the document into its logical structure is done during the import, since this task depends on the given input format, e.g., XML, Portable Document Format (PDF).

[7] https://dkpro.github.io/, accessed on September 3, 2018

Tokenizer

Tokenization is the process of dissecting a string of characters into fine granular sections of e.g., words and characters. These tokens serve as input for subsequent processing steps.

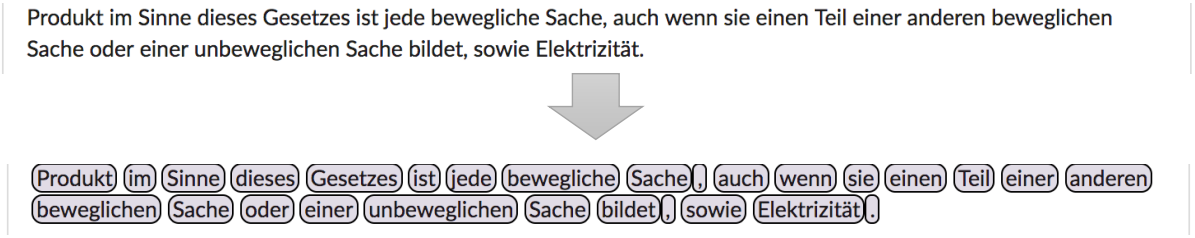


Figure 3.10.: An example of tokenizing article 2 of the product liability act.

The term token actually has a rather technical meaning within NLP. It can be a word, single characters, or a group of characters that logically belong together, such as enumeration items, e.g., "(a)", "i)", or "1)". Our implementation consistently treats words (including compound nouns such as "Produkthaftung") as tokens. Remaining characters, such as enumeration items or punctuation marks, are treated as tokens as well.

There are different strategies to split a string into words. Our implementation supports the usage of context-free grammars, e.g., regular expressions.

Stemmer

Once the tokens are extracted from the documents, the stream of characters contains words, and other tokens, e.g., punctuation marks. Natural language is characterized by the phenomenon that words can be inflected, depending on their meaning within the context and grammatical usage. Stemming performs the reduction of a word into its root stem (see Lovins 1968). Thereby, different strategies exist to reduce a word to its stem.

$$\begin{aligned} \text{Produkt}\underline{\text{s}} &\mapsto \text{Produkt} \\ \text{Hersteller}\underline{\text{s}} &\mapsto \text{Hersteller} \\ \text{Produkt}\underline{\text{e}} &\mapsto \text{Produkt} \\ \text{Entschädigung}\underline{\text{en}} &\mapsto \text{Entschädigung} \end{aligned} \tag{3.1}$$

In Listing 3.2 different stems of a word are shown, which have automatically been detected. The first two lines show the mapping of a word in the linguistic case genitive to its nominative. The remaining two lines show the mapping of two words in plural form to their singular form.

Stemming can usually be performed very quickly and does not require the attachment of additional knowledge sources, such as databases or dictionaries. Naive solutions are so-called suffix-stripping algorithms, that remove word ends, such as "s", "e", "en", etc. However, these operations can be performed very quickly and efficient, they come along with a limited functionality of reducing a word to its stem, e.g., "better" (see Hull 1996). To address these limitations more elaborated techniques are required, which are implemented in the so-called lemmatizer.

Lemmatizer

After the reduction of a word to its stem, the task of lemmatization ensues. Thereby, a word is not only reduced to its base form, but the context of the appearance of a word is also taken into account, which allows for more advanced reduction. For example, a stemmer cannot reduce the word "better" to "good", since this would required a dictionary look-up.

$$\begin{aligned} \text{aufgehoben} &\mapsto \text{aufheben} \\ \text{besser} &\mapsto \text{gut} \\ \text{besserer} &\mapsto \text{gut} \\ \text{Geschädigten} &\mapsto \text{Geschädigter} \end{aligned} \tag{3.2}$$

In Listing 3.2, different lemmas of a word are shown, which have automatically been detected. Each of them is correct. Note, that the lemmatizer recognizes the occurrence of the word "besser" (engl. "better") and maps it to "gut" (engl. "good"). This mapping could not be performed by looking at the given (inflected) form of the word.

During the extraction of the lemma for a given token, modern implementations of lemmatizers provide additional information about a word and its linguistic features, such as

- gender information (i.e., masculine, feminine, neutrum),
- part-of-speech (e.g., noun, verb, adjective), and
- case (i.e., nominative, genitive, dative, accusative).

Stemmer and lemmatizer are widely used in processing search queries to narrow or broaden the search queries. These operations are known as search query expansion and search query refinement (see Peng et al. 2007).

Subject tagger

We have developed an additional tagger, which is tailored to German legal documents and especially to statutory texts (e.g., laws), to extract the subject of a sentence. In general, the detection of the subject of a sentence or phrase is not trivial, computationally complex, and prone to errors. From a linguistic point of view, the subject of a sentence has to fulfill particular grammatical requirements. Within the German language, simple parameters exist that indicate the subject of a given sentence or auxiliary sentence.

One of these requirements is that the subject is a noun that has to be present in the linguistic nominative case to be considered as a noun. This linguistic phenomenon is known as the "Nominativkasus" and is widely accepted for the German language (see Dudenredaktion 2013). Especially in within a particular set of sentences, e.g., those not containing auxiliary sentences, it serves as a fast (but not very reliable) indicator for the subject of a sentence.

More reliable components to detect the subject of a phrase or a sentence are so-called dependency

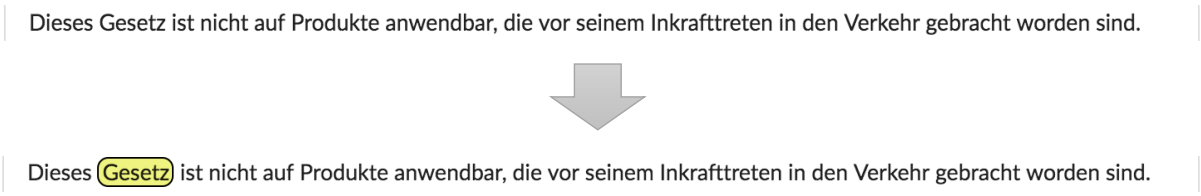


Figure 3.11.: An example of subject tagging according to the "Nominativkasus" rule.

or constituency parsers. The functionality of a dependency parser is introduced within this Section (see below).

Token	**POS tag**	**Case**	**Gender**	**Subject**
Gesetz	noun	nominative	neuter	yes
Produkte	noun	accusative	neuter	no
Inkrafttreten	noun	dative	neuter	no
Verkehr	noun	accusative	masculine	no

Table 3.2.: Four nouns with their automatically extracted linguistic features.

An exemplary extraction of the subject is shown in Figure 3.11. Within this sentence four different nouns exist: "Gesetz", "Produkte", "Inkrafttreten", and "Verkehr". The automatically extracted features, in this case the POS tag, linguistic case, and gender, for these four nouns are shown in Table 3.2. The functionality of this simple decision structure is implemented in the subject tagger component.

POS tagger

The POS tagger is a common component in computational linguistics. The POS tagger analyzes a stream of words within a given context, e.g., a sentence, and assigns the information on the part-of-speech to them. Different implementations vary in the level of detail to which the POS tags are assigned. The STTS[8] as well as the Penn Treebank[9] are the most commonly used POS tag sets.

Figure 3.12 shows an example of POS-tagged sentences from the German product liability act, in which the different words are highlighted according to the assigned POS tag. The screenshot highlights nouns in blue, cardinal numbers in dark green, verbs in gray, articles in light green, adverbs and adjectives in cyan. These assignments are internally handled as annotations (see Section 3.3).

[8] http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-table.html, accessed on September 3, 2018

[9] https://www.anc.org/penn.html, accessed on September 3, 2018

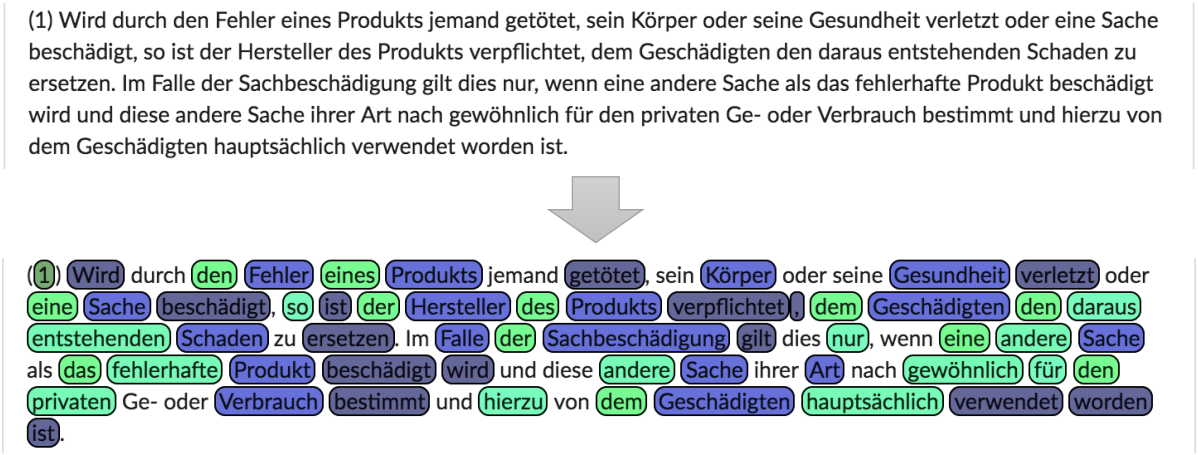


Figure 3.12.: An example of POS-tagging article 1 of the product liability act.

Metric Calculation Component

Beside the software components to annotate the text and to parse the natural language, the system provides functionalities to measure the lexical and semantic properties of each document that is imported into the data storage. These measures can be used to explore large datasets and to retrieve quantitative information, e.g., via distant reading (see Moretti (2013)).

Currently, the system supports the measurement of nine different quantitative indicators:

Name	Abbreviation	Indicated Complexity
Paragraph Count	# §	Linguistic & Structural
Sentence Count	# S	Linguistic & Structural
Word Count	# W	Linguistic & Structural
Structural Depth	D	Structural
Number of Outgoing References (internal)	INT	Structural
Number of Outgoing References (external)	EXT	Structural
Vocabulary Variety	V	Linguistic
Indeterminacy	I	Linguistic
Readability (Flesch-Reading-Ease)	FRE	Linguistic

Table 3.3.: Quantitative indicators for the complexity of legal texts as presented in Waltl and Matthes (2014).

The automated determination of the indicator shown in Section 3.3 allows the quantitative analysis of legal documents on the micro, meso, and macro levels. Micro-level inspection is the analysis of one particular document, e.g., the readability for the consolidated version of the German Civil Code. Meso-level inspection summarizes the analysis of a set of documents that form a logical group, e.g., all judgments referring to one particular article. Finally, the macro level contains the analysis of all available documents of a given type, e.g., all German laws.

Named Entity Recognizer

The extraction of so-called "named entities" is a common task in NLP. Different named entities can occur in written texts. These can also vary throughout different domains. Most commonly accepted are the following different categories:

1. Persons
2. Organization
3. Locations
4. Expressions of time
5. Quantities
6. Monetary values
7. Percentages

This list can slightly change over time with respect to the documents to be analyzed and the use cases that are going to be addressed. For the particular domain of legal texts, references (links or citations) to other legal documents could be considered as named entities. These are typical for the legal domain, but hardly occur in written texts in general. The same argument holds for the types of particular job titles. Within legal documents it might be relevant to not only extract a person's name, but also their job title, such as lawyer, judge, CEO, secretary.

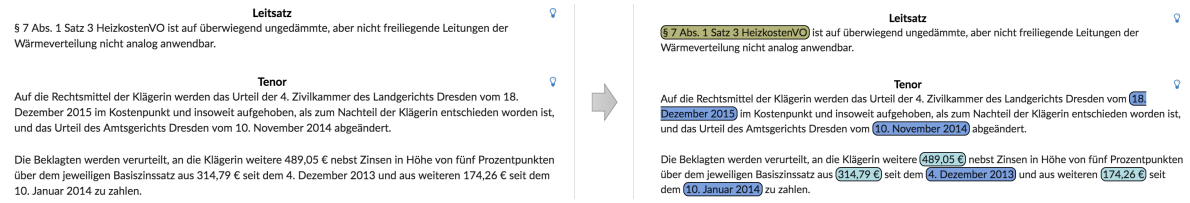


Figure 3.13.: An example of automatically recognized named entities in a German judgment.

Figure 3.13 shows the example of an annotated German judgment. On the screenshot three different types of annotations are recognized and highlighted: references (brown), expression of time (dark blue), and monetary values (light blue).

In general, four different strategies exist to automatically extract named entities from texts (Glaser 2017):

Rule-based: The most straight-forward implementation is to create rules that can be executed by software and that allow systems to determine named entities. These rules might be expressed as regular expressions, as Apache Rule-based text annotation (Ruta)s, or by any other declarative expression language.

Template-based: A given text is compared with a template, in which every occurrence of a named entity is filled with a placeholder. Algorithms compare the given text with the template and can extract the differences between the two documents. These differences are the named entities.

This is a trivial, but very effective measure to extract named entities. However, it relies on the fact, that one can provide a template representing the document with placeholders. This holds especially for documents that are highly-standardized, e.g., forms of rental contracts, tables, as their layout structure can be used to extract named entities.

Knowledge-based: Due to the emergence of publicly and freely accessible data, e.g., Wiktionary[10] and Wikipedia[11], the recourse on these data sources can be used for named entity recognition. These knowledge bases can be queried in order to determine whether a text contains a named entity that is described in an external data source or not. The hypothesis behind this strategy the assumption that if an entity is mentioned and described in this external source, it is most likely to be a candidate for a named entity, e.g., locations, persons, companies.

Since the mentioned data sources are not only dictionaries or a flat list of lexical units, but ontologies with an internal structure, logic, and semantic relationships, every record provides insights about dependencies and its semantic type.

Typical implementations, such as DBpedia Spotlight by Mendes et al. (2011), perform four different steps during the annotation: i) spotting, ii) candidate selection, iii) disambiguation, and iv) annotation.

Machine Learning: The fourth method to extract named entities from documents is based on Machine Learning (ML) technology. Within this framework, three different methods are common: i) supervised ML, ii) semi-supervised ML, and iii) unsupervised ML.

Supervised ML (SML) approaches require a large annotated corpus to train internal algorithms, such as conditional random fields (CRF), decision trees (DT), support vector machines (SVM). The training data is analyzed to memorize a list of entities. This list is transformed into an internal representation to detect such entities based on a set of distinctive and discriminative features. The resulting decision structures are subsequently applied to unknown documents (test data) in order to extract similar entities.

Semi-supervised ML (SSML) approaches start the learning process with a smaller set of training data (bootstrapping). During the learning phase, the algorithm tries to identify named entities, but retrieves user feedback steering the learning process. This is supposed to decrease the training effort and therefore lowers the size of a required training data set. The provision of a large and representative training set is still considered to be the main reason for the lack of adoption of semi-supervised ML (see Hastie et al. 2009).

Unsupervised ML (USML) is a general clustering task which relies on data analytics methods to identify patterns and regularities. USML Named Entity Recognition (NER) algorithms are focused on clustering entities regarding their content (covered text) and context. USML is rarely used for NER, as it cannot achieve the performance of SML or SSML (see Jurafsky and Martin 2014).

[10] https://www.wiktionary.org, accessed on September 3, 2018
[11] https://www.wikipedia.org, accessed on September 3, 2018

Dependency parser

More elaborated components to extract and assign roles to tokens, words, and phrases of legal texts require advanced grammatical information, such as relations between words and phrases. A common operation is the parsing of the dependency grammar (see Nivre 2005). The operation extracts the connections between words and phrases based on grammatical relations. This could for example be used to extract the subject and object of a sentence, or to collect connected phrases.

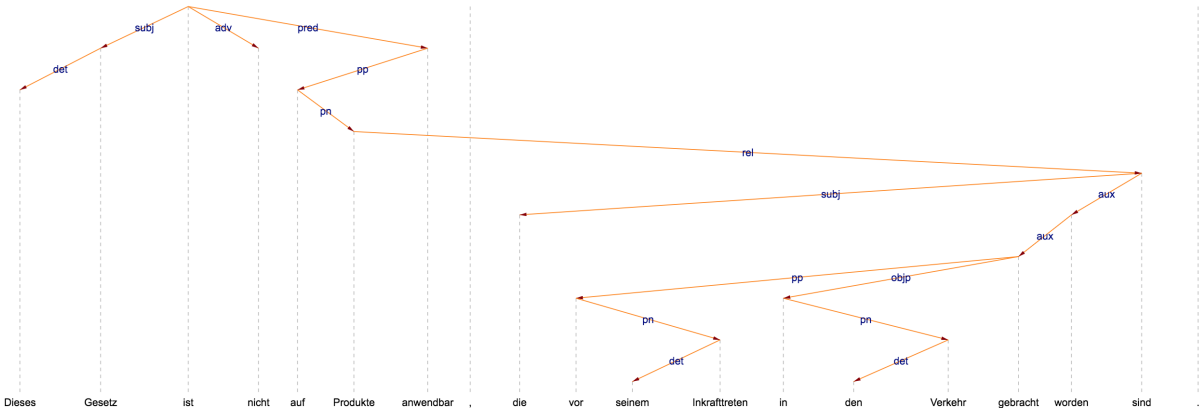


Figure 3.14.: Automatically extracted dependency grammar of a German legal sentence[12].

Figure 3.14 shows the example of a parsed sentence and the extracted dependency relations between the linguistic units, i.e., tokens. The parser used for the demonstration is an open source software and freely accessible (see Sennrich et al. 2013 or Nivre et al. 2006). The visualization shows dependencies between tokens and phrases. It extracts relations like "subj" (subject), "det" (determiner), "pred" (predicate), and auxiliary sentences.

The extraction of this information is complex from a computational point of view. Software components extracting this type of information typically require a lot of resources, e.g., processing time and memory. In addition, the general extraction quality of these grammatical relations for sentences in the German language is rather low. Highly specialized taggers are required to reach the necessary quality for the legal domain. For a small dataset from the legal domain this has been achieved by Sugisaki (2017).

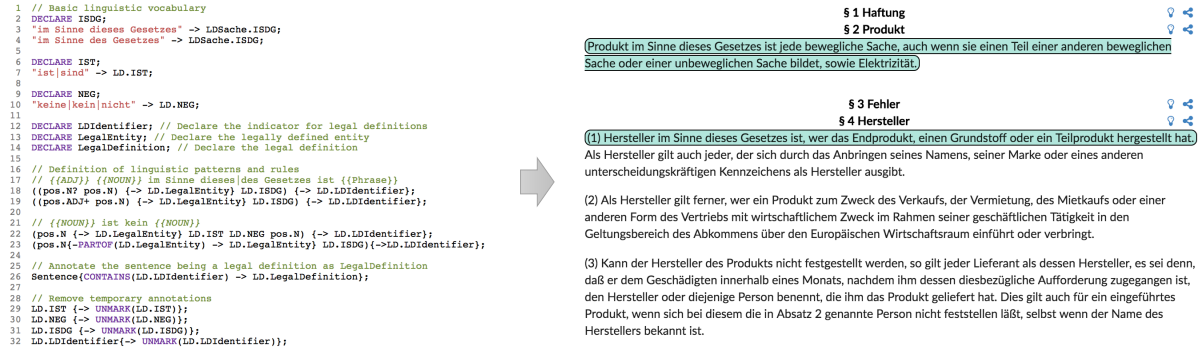
Complex Pattern Recognizer

To extract complex linguistic patterns, while reusing the linguistic information of components integrated into the text analysis engine, it was necessary to implement a pattern recognition component, that is easy to integrate and extend. The used software architecture allows for the simple integration of an existing open-source rule-based information extraction engine, namely Ruta.

Ruta can easily be integrated into UIMA and represents an imperative rule language where a rule combines a pattern of annotations; see Klügl et al. (2016). The Ruta workbench is an

[12]Visualization created using https://pub.cl.uzh.ch/demo/parzu/, accessed on September 3, 2018

Eclipse plugin which provides editing support with syntax highlighting and further features like rule explanation, rule validation, and the automatic creation of the descriptors based on the script containing the rules.



```
// Basic linguistic vocabulary
DECLARE ISDG;
"im Sinne dieses Gesetzes" -> LDSache.ISDG;
"im Sinne des Gesetzes" -> LDSache.ISDG;

DECLARE IST;
"ist|sind" -> LD.IST;

DECLARE NEG;
"keine|kein|nicht" -> LD.NEG;

DECLARE LDIdentifier; // Declare the indicator for legal definitions
DECLARE LegalEntity; // Declare the legally defined entity
DECLARE LegalDefinition; // Declare the legal definition

// Definition of linguistic patterns and rules
// {{ADJ}} {{NOUN}} im Sinne dieses|des Gesetzes ist {{Phrase}}
((pos.N? pos.N) {-> LD.LegalEntity} LD.ISDG) {-> LD.LDIdentifier};
((pos.ADJ+ pos.N) {-> LD.LegalEntity} LD.ISDG) {-> LD.LDIdentifier};

// {{NOUN}} ist kein {{NOUN}}
(pos.N {-> LD.LegalEntity} LD.IST LD.NEG pos.N) {-> LD.LDIdentifier};
(pos.N{-PARTOF(LD.LegalEntity) -> LD.LegalEntity} LD.ISDG){->LD.LDIdentifier};

// Annotate the sentence being a legal definition as LegalDefinition
Sentence{CONTAINS(LD.LDIdentifier) -> LD.LegalDefinition};

// Remove temporary annotations
LD.IST {-> UNMARK(LD.IST)};
LD.NEG {-> UNMARK(LD.NEG)};
LD.ISDG {-> UNMARK(LD.ISDG)};
LD.LDIdentifier{-> UNMARK(LD.LDIdentifier)};
```

§ 1 Haftung
§ 2 Produkt
Produkt im Sinne dieses Gesetzes ist jede bewegliche Sache, auch wenn sie einen Teil einer anderen beweglichen Sache oder einer unbeweglichen Sache bildet, sowie Elektrizität.

§ 3 Fehler
§ 4 Hersteller
(1) Hersteller im Sinne dieses Gesetzes ist, wer das Endprodukt, einen Grundstoff oder ein Teilprodukt hergestellt hat. Als Hersteller gilt auch jeder, der sich durch das Anbringen seines Namens, seiner Marke oder eines anderen unterscheidungskräftigen Kennzeichens als Hersteller ausgibt.

(2) Als Hersteller gilt ferner, wer ein Produkt zum Zweck des Verkaufs, der Vermietung, des Mietkaufs oder einer anderen Form des Vertriebs mit wirtschaftlichem Zweck im Rahmen seiner geschäftlichen Tätigkeit in den Geltungsbereich des Abkommens über den Europäischen Wirtschaftsraum einführt oder verbringt.

(3) Kann der Hersteller des Produkts nicht festgestellt werden, so gilt jeder Lieferant als dessen Hersteller, es sei denn, daß er dem Geschädigten innerhalb eines Monats, nachdem ihm dessen diesbezügliche Aufforderung zugegangen ist, den Hersteller oder diejenige Person benennt, die ihm das Produkt geliefert hat. Dies gilt auch für ein eingeführtes Produkt, wenn sich bei diesem die in Absatz 2 genannte Person nicht feststellen läßt, selbst wenn der Name des Herstellers bekannt ist.

Figure 3.15.: Ruta script, that matches the specified named or semantic entities (e.g., legal definitions) based on pattern matching when applied to a law text (see Waltl et al. 2016).

Figure 3.15 shows an exemplary Ruta script and its application to the German product liability act. The script formalizes the language so it can be parsed and executed automatically, i.e., a complex pattern recognizer (see Klügl et al. 2016). This method of extracting information is called rule-based information extraction and is still predominantly used in the industry, as a recent study by Chiticariu et al. (2013) has shown.

Pros	**Cons**
Declarative	Heuristics
Easy to comprehend	Requires tedious manual labor
Easy to maintain	
Easy to incorporate domain knowledge	
Easy to trace and fix the cause of errors	

Table 3.4.: Pros and cons of rule-based information extraction from a survey among 54 different software vendors, as summarized by Chiticariu et al. (2013).

The findings of Chiticariu et al. (2013) are summarized in Table 3.4. As of 2017, rule-based information is not en vogue any longer, since academic research is investigating machine-learning based approaches. However, the authors state that the "academic NLP community needs to stop treating rule-based IE [Information Extraction] as a dead-end technology" (Chiticariu et al., 2013, p. 830), since 67% of the large software vendors are still purely working with rule-based information extraction.

The UIMA Ruta and its role within the UIMA framework is described in Section 3.5.3.4.

Implementation as UIMA Annotators

The different components of the text analysis engine are implemented in the programming language Java and run at the server side of the web application. Their implementation follows the design principle of modularity and interchangeability, which is fostered by the underlying architecture of Apache UIMA.

Every component, except the indicator calculation component, consists of a single (or multiple) UIMA annotator(s)[13]. The concrete implementation and relevant Java code are discussed in Section 5.

The systems have UIMA annotators to integrate the following components:

JudgmentReferenceAnnotator and LawReferenceAnnotator: Based on regular expressions citations and references are detected within legal documents. Since the lexical occurrence of these references vary within different types of legal documents, e.g., laws and judgments, these components handle the document types individually.

LeitsatzAnnotator: Given a particular judgment from a German court, this annotator extracts the paragraphs indicating the "Leitsatz" (engl. guiding principle) of a judgment, based on structural features. This can be considered as a short summary and contains the essence of a judgment. The "Leitsatz" is created by the judges and is part of the official document.

StopwordAnnotator: Based on a given dictionary, this component marks German stop words. Stop word removal is a basic operation in information retrieval, but needs to be applied carefully, since the occurrence of a stop word can make a huge difference for the semantics of a legal text, e.g., not, and, or.

ZitatAnnotator: Direct citations can be a valuable sources of information. Although these hardly appear in laws, judgments tend to cite the content of documents to which they refer, e.g., existing contract clauses, statements by witnesses, etc. These are indicated using different tokens, such as quotation marks. The extraction of this citations is done with this annotator.

SubjectAnnotator: The subject annotator is implemented as described in this section and the linguistic phenomena of "Nominativkasus" is implemented using straight-forward Java conditionals.

The annotators introduce different and more or less complicated functionalities. Each annotator handles exactly one particular function and contributes this to the overall annotation set. Based on their occurrence within the processing chain, they can reuse the (intermediate) results of a prior annotator. The concrete functionality of the Pipes & Filters is discussed in Section 3.5.

Furthermore, this section describes the different modules and how they are logically separated from each other. It also provides insights into the basic structure of the text analysis engine and how it could be extended if additional functionality for the semantic analysis is required.

[13]https://uima.apache.org/doc-uima-annotator.html, accessed on September 3, 2018

3.4.2. Active Machine Learning Classifier

In addition to the aforementioned software components that are mainly integrated as UIMA annotators, we have also integrated a separate service, which is particularly designed to classify named (legal) entities: an Active Machine Learning classifier.

The mentioned components, such as the complex pattern recognizer, dependency parser, or named entity recognition component, cannot update their linguistic model during the usage of the system. However, it is compelling to have a component that does not only creates annotations based on a pre-trained model, but "learns" during the usage of the system and while creating and deleting components. This is a suitable use case for supervised ML.

The component implemented uses a so-called AML technique to learn and adapt when the system is used. The component has been designed and trained to determine legal entities from statutory texts, such as obligations, prohibitions, legal definitions, etc. By incorporating the immediate feedback from the user, whether the prediction was right or wrong, it extends the existing functionality of the rather static and ex-ante specified and trained text analysis engine that cannot adapt and refine its models.

ML, AML, the used framework and its conceptual and architectural integration are discussed in Section 3.5.5.

3.5. Software Architecture for Processing Legal Documents

This section describes the design decisions made to implement a software architecture for semantically processing legal documents. Based on the consideration in Foundations and Related Work (Section 2) and the previous Sections 3.4.1 - 3.4, we assessed existing frameworks for the processing of legal documents, the result of which are summarized in Section 3.5.1. The processing architecture is described in detail in Section 3.5.2). We decided to reuse and adapt the existing UIMA framework. The consequences on the analysis of legal documents are discussed in Section 3.5.3.

For the implementation of the Active Machine Learning framework, we have chosen the same procedure. In Section 3.5.4, we will elaborate on the outcome of an assessment of potential candidates for our implementation. We discuss the general architecture in of the active machine learning framework in Section 3.5.5. Finally, we will introduce the selected machine learning framework Apache Spark in more detail (see Section 3.5.5.3).

3.5.1. Assessment of Processing Frameworks

In order to compare different processing frameworks on which a solution tailored to legal documents can be implemented, we have had a few basic requirements:

1. The framework shall support its usage within a web application.
2. The framework shall support the parallel processing of documents (thread-safety).

3. The architecture of the framework shall foster the reuse of software components.
4. The processing engine of the frameworks shall use a standardized data format and structure.

Based on these basic considerations, Tobias Waltl (2015, pp. 29) conducted an extensive study of existing software frameworks. Different frameworks were compared to each other. Taking this into account, we have drawn our conclusion for the analysis framework on which we have built our system.

3.5.1.1. An Overview of Natural Language Processing Frameworks

Apache UIMA: The concept of UIMA was developed by IBM with the main objective of providing means to foster the reuse of components for linguistic software components (see Ferrucci et al. (2009)). In 2006, IBM donated it to the Apache Foundation and since 2010, UIMA has been a top-level Apache project[14]. The large UIMA ecosystem provides a variety of features supporting language engineering, e.g., Eclipse plugins. The implementation of the core is available in Java and in C++ .

The most central concept of UIMA is the so-called Common Analysis System (CAS) and it follows a Pipes & Filters architecture (see Section 3.5.2). Throughout all the filters the central information repository, i.e., CAS, keeps all the linguistic information about a document. This object is instantiated once and is given to all components of a processing pipeline. Consequently, the components cannot directly communicate with each other. They retrieve their input from the CAS and the previous components and reinsert their determined output into the CAS again. The main objects within the CAS are annotations. The CAS can be accessed to utilize the information extracted along the pipeline. For the Java implementation, this object is call JCas object and enables efficient and programmatic access (see Gotz and Suhre 2004).

The core is a robust stand-alone component, which supports the usage within a multi-thread environment. It is not only suitable for usage in web applications, but allows the parallel processing of large document collections.

A considerable variety of software components exists that can easily be integrated into the pipeline model. A well-known and active repository is DKPro[15], which is mainly maintained by the research group of the Ubiquitous Knowledge Processing Lab[16] (UKP) from the Technical University of Darmstadt.

For rule-based information extraction based on Common Pattern Specification Language (CPSL), UIMA provides the so-called Ruta engine. Ruta consists of an imperative rule language where a rule combines a pattern of annotations and further conditions. Based on these conditions, actions are triggered which are executed if a given formal pattern

[14]https://uima.apache.org/, accessed on September 3, 2018
[15]https://dkpro.github.io/, accessed on September 3, 2018
[16]https://www.ukp.tu-darmstadt.de/ukp-home/, accessed on September 3, 2018

matches; see Klügl et al. (2016). More information about the UIMA Ruta is provided in Section 3.5.3.4.

GATE: Since 1995, the GATE has been developed by researchers and affiliates from the University of Sheffield. It is a widely used and mature software framework based on a TIPSTER architecture (see below). The overall objective is the provision of an infrastructure for the seamless integration of different NLP components. Its current version is 8.4.1[17] and is still maintained by an active community[18]. It offers a mature IDE, a large collection of compatible components, and a framework, which is called GATE embedded, allowing the integration of these GATE components into a Java application utilizing them (see Cunningham et al. 2011).

The architecture consists of three major elements:

1. GATE document manager (GDM)
2. Collection of reusable objects for language engineering (CREOLE)
3. GATE graphical interface (CGI)

The data model of the GDM is based on the TIPSTER and supports bulk processing large collections of documents, i.e., corpora. The data model handles both: text and annotations. The GDM is the central information repository and the component's common interface for reading and writing linguistic information. Documents are processed subsequently. The linguistic components are de-coupled, which means, that they do not exchange information directly with each other. Information is shared via the data model managed by the GDM. The CREOLE is the repository of NLP components. Within the GATE framework, these are called processing resources.

Another very interesting functionality of GATE is the rule-based information extraction engine based on CPSL. This follows from its ancestor TIPSTER. The engine is called Java Annotation Patterns Engine (JAPE) and enables the specification of linguistic patterns which are applied to a document corpus. Matches of these pattern definitions cause actions, e.g., creating an annotation, or calling another operation or function.

GATE is a powerful framework that is widely used in software-supported analysis of textual documents and NLP. It has an active user and developer community.

TIPSTER: An early architecture for processing of large document collections was proposed by Grishman (1996). The concept of annotating documents was implemented as a combination of stand-off annotating and enhancing documents with attributes. The system was developed in C, Common Lisp, and TCL. The system was not natively designed to be used in an environment that enables collaboration, e.g., a web application.

Although TIPSTER is not longer available the basic ideas of document-centric processing and of an efficient and lean implementation have been reused in modern system architectures, such as Ellogon, LIMA, or GATE.

[17] Released on June 9, 2017
[18] https://gate.ac.uk/, accessed on September 3, 2018

Ellogon: Ellogon[19] (see Petasis et al. (2002)) has been developed based on TIPSTERs architecture. Its core was written in C and handles the storage of textual data as well as the associated linguistic information, i.e., annotations. Annotations have four different attributes, namely an id, a type, a set of spans (denoting the annotated textual data), and a set of attributes (containing the linguistic information). Ellogon allows the reuse of software components during complex linguistic analysis. In addition, it has a graphical user interface to view configurations and results of processing pipelines.

Ellogon is still actively used by Greek research groups, for example by Katakis et al. (2016).

LIMA: Just like Ellogon, LIMA (Libre Multilingual Analyzer) is based on a TIPSTER-like architecture. It is primarily designed to support multilingualism, diversity of applications, extensibility, and efficiency. LIMA stores configurations of pipelines, processing units, resource definitions, and logging information in XML. It fosters the reuse of components within processing pipelines. Parallel processing of documents is not yet implemented, but was identified as a project for further research in Besancon et al. (2010). LIMA offers a broad variety of tools to test and evaluate linguistics modules.

There is an active community supporting and developing LIMA, mainly working at the CEA LIST[20].

Whiteboard architecture: The integration of heterogeneous components for NLP was the main design goal for the Whiteboard architecture. It was originally designed in 1994 and consists of a central instance, the coordinator, that orchestrates the different components via so-called managers. These managers encapsulate the components and "hide and transform" their functionalities (see Boitet and Seligman 1994).

This architecture was essentially useful when hardly any standardized format of data exchange between software components was established.

TALISMAN: In 1995, a multi-agent architecture for NLP has been proposed by Stefanini and Demazeau (1995). They argue, that the sequential architecture does not allow the real exchange between different software components. Based on the analysis of sentences they showed the applicability of their approach using distributed artificial intelligence.

TALISMAN was inspired by the idea of distributed artificial intelligence, which was very popular in the 90s. Thereby, no central authority orchestrates the data exchange between components, as they are self-organized. However, the recent years have shown the advantages of well-defined unidirectional processing pipelines in terms of performance and modularity.

TalLab: Based on the need for a software architecture aiming to "ease the work of software engineers producing, deploying and monitoring the [NLP-based on-line] applications" the TalLab architecture was developed (see Wolinski et al. 1998). In the main publication, four requirements have been listed that were guiding their implementations: i) Providing malleability to facilitate the evolution of applications, ii) maximizing openness to take ad-

[19] http://www.ellogon.org/, accessed on September 3, 2018
[20] http://www-list.cea.fr/en/, accessed on September 3, 2018

vantage of foreign components, iii) increasing efficiency to extend the scope of applications, and iv) ensuring exploitability to guarantee the integration of applications.

Similar to TALISMAN, TalLab was implemented "an OS-based multi-agents system" that sends messages between the agents to inductively process a natural language based text. And just like TALISMAN, these agent-based architectures have not been properly accepted by the linguistic community. The creation of powerful sequential pipelines has distinct advantages over a self-organized multi-agent system.

Heart of Gold: The Whiteboard architecture has been improved by introducing an additional hybrid architecture called "Heart of Gold". Heart of Gold is a middleware architecture that "generalizes WHITEBOARD into various dimensions such as configurability, multilinguality and flexible processing strategies" (see Schaefer 2007, p. 3). It consists of three main components, namely a modules package, an XSLT service, and a module communication manager. The modules perform actual NLP tasks, or encapsulate external services, the XSLT service handles the exchange and mapping between the modules, whereas the module communication manager orchestrates the pipelines, etc.

The annotations made by a module are stored as XML. The schema of the XML has not been standardized, i.e., no DTD. The developers chose a light-weight XML format supporting the "Robust Minimal Recursion Semantic" (see Copestake (2007)).

As stated above, an in-depth analysis of the different frameworks and their suitability was carried out in Waltl (2015). Within this work, we will limit ourselves to present the aggregated outcome.

3.5.1.2. Conclusion

In a recent book, Wilcock (2017) investigated the capabilities from a pure computational linguistic perspective and concluded that the most promising software frameworks for analyzing textual documents are UIMA and GATE. For a state-of-the-art NLP framework, he emphasized the importance of four different criteria:

1. pipeline configuration,
2. platform independence,
3. graphical interface, and
4. stand-off XML mark up within modern linguistic annotation frameworks.

Wilcock states: "These developments [...] were successfully integrated into GATE and UIMA, the main large-scale modern annotation frameworks.". These results are in line with our independent investigations and analyses, as summarized in Table 3.5. In addition to the four different criteria, we have added another four dimensions, with particular focus on a flexible type system, as this changes throughout the analysis of different document types and use cases. From a software engineering perspective, we have also added the usage in a multi-threaded environment, e.g., a web application.

	TIPSTER	Ellogon	LIMA	Whiteboard Architecture	TALISMAN	TalLab	Heart of Gold	GATE	UIMA
Web application	-	-	-	-	-	-	++	++	+++
Reuse of components	-	+++	++	++	+	-	++	+++	+++
Type system	++	-	-	-	-	-	+	-	++
Common data format	+++	+++	+++	-	-	-	-	+++	+++
Integration and Interchangeability (i.e., pipelines)	+	++	-	++	-	-	++	+++	+++
Parallel processing	-	-	-	+++	+++	+++	+++	-	+++
Framework	-	+++	++	-	-	-	++	+++	+++
Community	-	++	+	-	-	-	-	+++	+++

Table 3.5.: Framework comparison considering ten different language processing frameworks and eight different categories.

The table lists the different frameworks (columns) and the evaluation criteria (rows). Each cell contains the result of the degree of fulfillment, whereas '-' corresponds to "not fulfilled or not applicable" and '+++' corresponds to the highest degree of fulfillment.

As shown in Table 3.5 – based on our criteria and objectives – the Apache UIMA has the highest degree of fulfillment. Consequently, we have chosen the Apache UIMA for our implementations.

The next two Sections, 3.5.2 and 3.5.3, provide more details about the conceptual structure behind Apache UIMA and the concept of Pipes & Filters, and their importance during the analysis of legal documents.

3.5.2. Pipes & Filters Architecture

In software engineering, numerous different software architectures have been developed to support different needs and purposes. A very common architecture to perform the analysis of data and data streams is the so-called Pipes & Filters architecture. A schematic overview of this architecture is shown in Figure 3.16.

Pipes & Filters architectures consist of two main components:

1. **Filters** are the components that perform the processing, i.e., processing step.
2. **Pipes** are the connectors between two consecutive processing steps.

Within Figure 3.16, filters are shown as triangles and pipes are shown as lines connecting these triangles. The data flow is from left to right and can be read straight-forward. The different components and their concrete processing tasks from Figure 3.16 are described in Section 3.4.1.4.

A (legal) document is selected and inserted into the first step of the pipeline, which is the *Splitter*. The *Splitter* divides the large document, e.g., the German Civil Code, into its sections. The set of sections is forwarded to the *Segmenter*, which breaks up each section into a smaller subject of analysis. This set is then forwarded into a *Tokenizer*, which separates the individual tokens, e.g., words, punctuation, brackets, from each other.

After this stage, the large initial document has been disassembled into small portions of in-

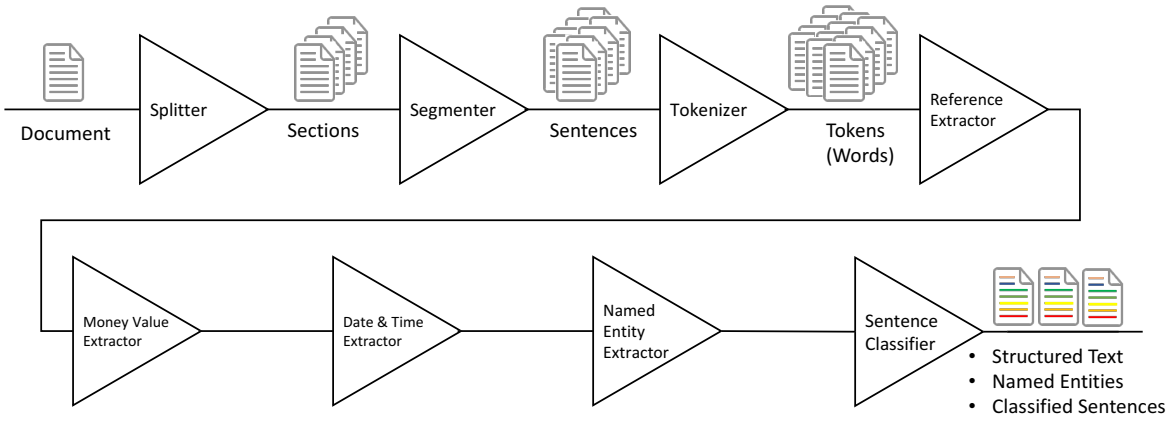


Figure 3.16.: Schematic overview of a Pipes & Filters architecture for the analysis of legal documents (extension of a pipeline model as introduced in Waltl et al. 2016).

formation, which can be analyzed individually or within their larger context. At this stage it becomes clear that each filter continuously adds new information to the pool of information objects, which consists of the document, sections, sentences, tokens, and POS tags. But this is not necessarily the case. Filters could also be used to reduce and decrease the information stored in the pool.

After the document has been split into fine granular pieces of information, different extraction components, such as a reference extractors, a money value extractor, a date & time value extractor, and a named entity extractor, are applied. It does not matter for the pipeline components which particular technology, e.g., context-free grammars, or machine-learning, is used within a component. This complexity is encapsulated. From this architectural perspective, the different components are black boxes.

Finally, a sentence classifier, e.g., a UIMA Ruta component or an active machine learning component, performs a matching on top of the stream of available information to assign semantic types.

This architecture has three main advantages that make it so attractive in the field of data processing:

1. **Simplicity:** The overall idea is rather simple to implement and does requires the large overhead of an "master module" orchestrating the whole process. Each filter is initiated by its predecessor and the data pipe in between. In addition, the system can easily be understood and analyzed by looking at the individual components.

2. **Independence:** Each filter works as a stand-alone component and does not (necessarily) depend on the performance of the other filters. Trivially, the results generated by a filter are in many cases reused by a subsequent component. For example the tokenizer could still extract the different tokens and words from a text even though the sentence splitter might have a very low precision rate. Consequently, to achieve a high overall accuracy, the accuracy of the individual components needs to be high as well.

3. **Modularity:** The different software components communicate over the pipes, which can be considered as well-defined interfaces and objects that are passed through the pipeline. Hence, the components can be replaced in their order or even by other more advanced or specialized components. For example, if a new document needs to be processed, e.g., a statutory text from another legal domain or even legal system, in which the sentence boundaries follows another logic, e.g., due to a large variety of enumerations and other non-sentence-breaking punctuations, it is sufficient to provide another sentence splitter whereas the remaining processing pipeline does not need to be changed.

Based on these considerations, the Pipes & Filters architecture is well-suited for a research prototype with the target to analyze legal documents. Especially when it serves as a platform for interdisciplinary data science that has to support the implementation of multiple use cases for semantic analysis. The system needs to be flexible enough to support different document types and formats. In addition, the platform needs to decrease the effort of adding new functionality when needed.

Limitations and Critical Remarks

In addition to the main advantages, the Pipes & Filters architecture also has several limitations and drawbacks:

1. **Architectural overhead:** The Pipes & Filters architecture introduces additional overhead in terms of implementation effort. Additional java classes and objects are required to define the structure of the pipeline and to add new filter classes that perform a desired behavior. The linguistic components are wrapped into a pre-defined structure of interfaces, so that those can be integrated easily and are compatible among each other.
2. **Complex configuration:** A pipeline needs to be configured. The sequence of the components needs to be specified ex ante. The different components also need some configuration in order to load the correct model, e.g., the language of the document. Depending on the implementation and the feature this could also be done during runtime. However, the configuration of many variables before pipeline execution adds additional complexity to the Pipes & Filters architecture and the system, which uses it for processing of documents.
3. **Dependencies within the pipeline:** A main advantage is the modularity of the components, which perform different tasks and produce different (intermediate) results that are either used by another component or as part of the final result. From this circumstance, a disadvantage also arises: the components can be dependent on each other. They are relying on the results that another component has produced beforehand. For example, the tokenizer in Figure 3.16 may rely on the sections, which are generated by the splitter component. Using the tokenizer without the splitter does not work, as the tokenizer does not get the input it requires. For simple pipelines that are statically pre-defined, this might not be a problem, however, if users that are not most familiar it could have difficulties during the parametrization.
4. **Linear pipelines:** As the pipelines have dependencies among the components, and to keep the complexity and execution manageable, the pipelines are executed sequentially

and do not allow for forks or concurrent execution of components. This is also, as we will later see in Section 3.5.3.1, due to the fact that robust implementations, such as the Apache UIMA, share one common data object to store and forward information along the pipeline. Managing the access to this object becomes challenging due to concurrent manipulation, especially if the pipeline is used in multi-threaded environments.

5. **Memory:** The dependencies among the components and the sequential usage of the intermediate results of a component require a common data object that is passed through each and every component. One of the main challenges is to keep the data object maintainable because it is usually designed to constantly grow during the pipeline execution phase. The continuous growth could be mitigated if a component cleared would the common data object. This would be possible from a technical point of view, however, since it does not know which operations are performed afterwards, data that is required in a subsequent phase might be deleted.

The limitations and drawbacks should be taken into account during the architectural decision-making process. The architecture that was reused within our system was the Apache UIMA, which was already briefly introduced in Section 3.5.1. The following Section elaborates in detail on the architecture, the type system, the pipeline model, the available (linguistic) components, and on how they can be reused and adapted for the legal domain.

3.5.3. Apache UIMA

The Apache UIMA was used as the base framework for our implementations. The UIMA project claims to be divided into three different main areas[21]:

1. Frameworks
2. Infrastructure
3. Components

"Frameworks" are different implementations which encapsulate the integration of the components within different environments, such as C++ or Java, or scale-out frameworks for their usage within large computing clusters. "Infrastructure" covers the provision of tools and work-benches to support the adoption and efficient integration into language engineering workflows. Finally, "components" describe the large collection and repository of reusable software components.

This section describes the main components of the Apache UIMA and provides a more detailed description of the pipeline model and its implementation within Apache UIMA.

[21] https://uima.apache.org/, accessed on September 3, 2018

3.5.3.1. Pipeline Model in Apache UIMA

Apache UIMA uses an advanced architecture to represent the Pipes & Filters architecture. Figure 3.17 depicts the resulting and instantiated workflow. It is built on the studies provided in Section 3.5.2.

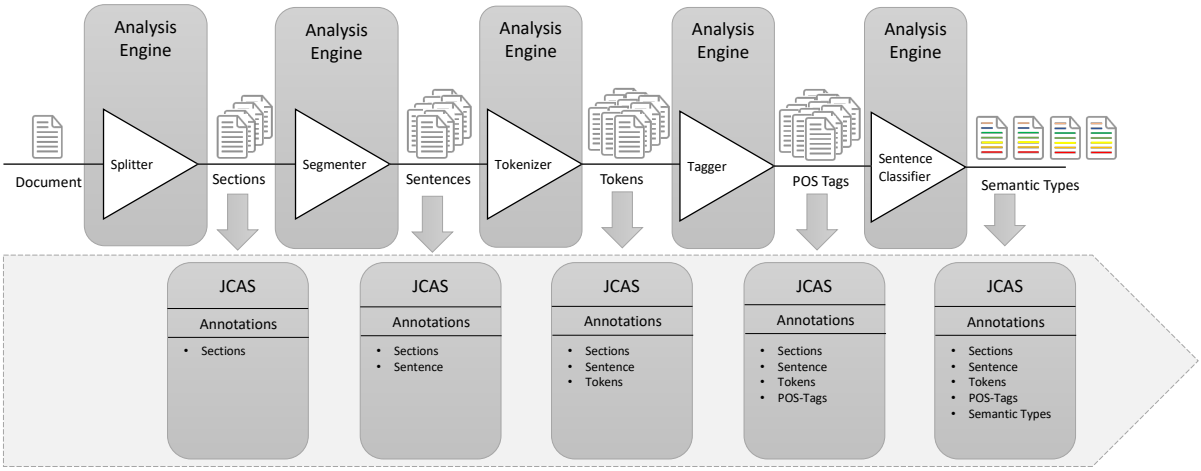


Figure 3.17.: An aggregated analysis engine with integrated JCAS object.

The processing consists of three main objects that are implemented as Java classes. The objects and their responsibilities during the process can be described as follows:

Analysis Engine: UIMA follows a component-based design principle. These components that are used for processing of information, e.g., text, are called Analysis Engines. Each Analysis Engine (AE) has an associated XML descriptor, which specifies the expected input and the produced output. For example, the POS tagger requires a set of annotations from the token type. The descriptor is tightly coupled with the AE. Changing the components input or output typically implies changes within the descriptor.

To handle these changes more efficiently, another abstraction layer was introduced, which is called called Apache uimaFIT (see Ogren and Bethard 2009). In this process, the AE are extended with Java annotations that deliver a detailed description of the in- and output of a component. Thus, with uimaFIT, it is possible to describe a component directly in its Java implementation and therefore, do not have to create and maintain the XML descriptors do no longer have to be created and maintained.

Interchangeability of two AE is given if the constraints regarding the in- and output requirements are met. Trivially, if a component needs to have a text, segmented into its sentences, it relies on the previous components to deliver these annotations. As the UIMA is strongly typed, i.e., each annotation type is defined as a separated Java class and not only as the description of some XML document, configuration file, or generic class, one can rely on these definitions and must not expect conflicting annotations with the same name. This is a common problem in other frameworks, e.g., GATE. More information about the UIMA type system is provided in Section 3.5.3.3.

Aggregate Analysis Engine: The overall object, handling the aggregation of the different analysis engines, is called the Aggregate Analysis Engine (AAE). The AAE handles the CAS object, by managing its complete life cycle. Its creation is initiated here, as well as the creation of new objects, i.e., annotations. It holds various meta-data for the processing pipeline, e.g., information about the order of the analysis components or the language information. It allows parallelization and so-called flow constraints, which specifies the routing of the CAS object during the processing. This is important to ensure concurrent usage within multi-threaded environments, e.g., web applications.

In principle, AAE can be executed in parallel. Physically, an AAE is executed single-threaded. For real multi-thread applications, UIMA Asynchronous Scaleout (AS)[22] is available. In addition to physical multi-threading, UIMA AS provides mechanisms allowing for scaling on a cluster of machines. Standard UIMA components are executed within UIMA AS without any code modifications. Static fields should be avoided, since these are handled by the Java class-loader and are shared among multi-threaded components. The UIMA AS creates multiple instances of UIMA components which ensures that each AE instance is accessed by exactly one thread.

CAS and JCAS: The CAS and its wrapper for the Java, called Java Cover Classes based Object-oriented CAS (JCAS), are the central information repositories within an AAE. The CAS is an object that enables AEs to access a the document and metadata about that document. Analysis components read from a CAS interface in order to perform their analyses and may write new metadata back to the CAS interface.

At the time of the instantiation, the CAS will be initialized with main information about a document, e.g., its content, and additional meta-data.

This section provided an introduction into the basic structure and involved components during the analysis of a document within the Apache UIMA. The next section will add information about available software components for the UIMA ecosystem that foster and accelerate the process of language engineering.

3.5.3.2. Software Components for Apache UIMA

Although it is rather easy to implement a new custom annotator for the Apache UIMA, modern software engineering heavily relies on the re-use of available software components. For the UIMA ecosystem, a large variety of software components has already been implemented. A well-known collection of these software components is the repository "DKPro" and was described in-depth by de Castilho and Gurevych (2014). The core of DKPro offers 94[23] different linguistic components, that are either available as under the permission of an Apache License model or under the GNU GPL.

[22] https://uima.apache.org/doc-uimaas-what.html, accessed on September 3, 2018
[23] Version 1.8.0, accessed on September 3, 2018

The repository[24] differentiates between 17 different categories of components for linguistic analysis:

1. Analytics components
2. Checker
3. Chunker
4. Coreference resolver
5. Language Identifier
6. Lemmatizer
7. Morphological analyzer
8. Named Entity Recognizer
9. Parser
10. Part-of-speech tagger
11. Phonetic Transcriptor
12. Segmenter
13. Semantic role labeler
14. Stemmer
15. Topic Model
16. Transformer
17. Other

The availability of these software components fosters the creation of applications with a particular focus on a domain, e.g., legal documents. Nevertheless, the domain adaptation, including the provision of specific components (see Section 3.4.1.4), still needs to be done.

3.5.3.3. UIMA Type System

The Apache UIMA provides a very generic type system that allows the specification of annotation types (see Section 3.2.1). These annotation types are called "types" and they support the specification of attributes so-called "features". The types follow a very strict naming convention, which is used to resolve the space of a type. This can be illustrated by the following two examples:

de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Token

The first example shows a built-in annotation type from the DKPro framework. The annotation

[24] https://dkpro.github.io/dkpro-core/releases/1.8.0/docs/component-reference.html, accessed on September 3, 2018

type is Token and its namespace is de.tudarmstadt.ukp.dkpro.core.api.segmentation.type. This is a common naming convention in software engineering, wherever different workspaces exist and a full qualifier to resolve this is required.

The attributes of the UIMA also follow this naming convention:

uima.tcas.AnnotationBase:sofa
uima.tcas.Annotation:begin
uima.tcas.Annotation:end

These three features exist in every type. They identify the "Subject of analysis" Sofa and the region of the annotation. According to their namespace, they are within the package uima.tcas and belong to the type Annotation, respectively AnnotationBase, and have the name sofa, begin, and end.

The UIMA type system describes how the JCAS internally handles, addresses, and manages the annotations. In our system, the complexity of internal and technical representation is reduced after a successful processing. The type systems structure is mapped onto the *Annotation* object of the data model that is kept within the local data storage.

The mapping of the UIMA type system preserves all types and features. However, our system adds more additional information to each annotation that is stored, e.g., *creationDate* and *id*.

3.5.3.4. UIMA Ruta (Rule-based text annotation)

In order to provide an annotation technology that is fully supported and integrated into Apache UIMA, we decided to integrate Ruta[25], which consists of two main components:

1. An analysis engine (see Section 3.5.3.1) interpreting and executing the rule-based scripting language.
2. A workbench providing end-user support for the development, maintenance, and evaluation of these rules.

The UIMA Ruta language is an imperative rule language extended by scripting elements (see Klügl et al. 2016). A rule consists of a pattern definition based on annotations and a set of conditions. If this pattern matches, the specified actions are executed. The actions are performed on the matching annotations.

A rule is composed of a sequence of rule elements and a rule element essentially consists of four parts:

1. A **matching condition** which is the basic condition that needs to be fulfilled such that the actions are executed. This basic conditions is usually evaluated in the text, or in a region of the text, or in a set of existing annotations, of a given document.
2. A **quantifier**, which is optional and allows the specification of multiplicities on the match-

[25]https://uima.apache.org/ruta.html, accessed on September 3, 2018

ing condition. It is possible to declare the conditions only to match if a given number of subsequent matches is fulfilled.

3. A **list of conditions** provides an additional way to provide conditions beside the *matching condition* that need to be fulfilled in order to to execute the actions.
4. A **list of actions** defines the actual operations that are performed if all prerequisites are met. In most cases, these specify the creation of new annotations or the modification of existing annotations.

In the following two examples, for simple Ruta rules are shown:

```
1 DECLARE SECTIONMARK;
2 DECLARE SECTIONNUMBER;
3
4 "§|§§" -> SECTIONMARK;
5
6 SECTIONMARK Token{REGEXP("[0-9]+[a-z]?") -> MARK(SECTIONNUMBER)};
```

The first rule marks each occurrence of the symbols '§' and '§§' in the text and annotates them with the type SECTIONMARK. The second rule identifies combinations of numbers and one optional character that occur immediately after a SECTIONMARK annotation. Thereby, the action MARK(SECTIONNUMBER) is only executed on the token that is matched by the regular expression, not for the SECTIONMARK annotation.

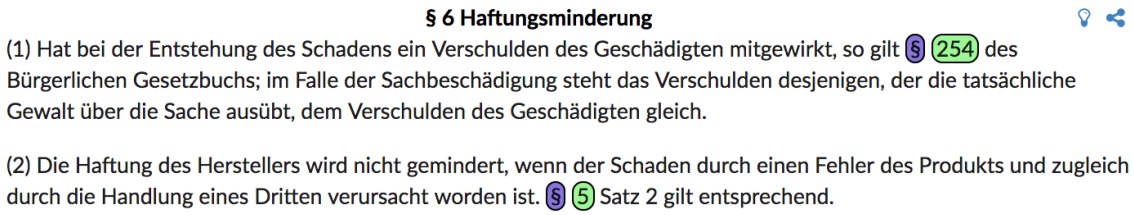


Figure 3.18.: An example of a highlighted text based on annotations by UIMA Ruta rules.

Executing this script snippet would annotate the section symbols and subsequent numbers, each of which with a different annotation type. The result is illustrated in Figure 3.18; another example was already shown in Figure 3.15.

It is important to mention that the UIMA Ruta allows to create annotation based on existing annotations and on linguistic features. It is fully integrated into the type system of Apache UIMA and allows to reuse the whole annotation functionality that is provided by any other component within the AAE.

3.5.3.5. UIMA and its Adoption in the Legal Domain

Although from a software technical point of view, Apache UIMA is the most advanced and mature framework for the semantic analysis of textual documents, its usage within the field of the analysis of legal documents has been rather modest.

A very promising — and to the best of our knowledge only — approach (beside ours) has been presented by Grabmair et al. (2015). They have re-used the UIMA type system to support the re-ranking of search results. Their approach was built on the analysis of vaccine injury decisions and the usage of UIMA Ruta and Weka[26] to extract and assign semantic types on the sentence and sub-sentence level of the legal texts. Grabmair et al. used the UIMA framework to extract legal concepts from the vaccine injury decisions. However, the focus of their research was the usage of the framework to extract information to re-rank search results. The information extraction component has not been the main focus of their published work. A detailed description of the software components, especially of the computational linguistic operations, that have been implemented and how they interact with each other is missing. The thesis at hand can be considered as a contribution to close exactly this particular gap of describing and proposing a generic software architecture that is powerful and modular and fosters the interoperability of results from different research groups.

It is to be hoped that more research groups, especially in the field of legal informatics, recognize the importance of performing data analysis on a common technological platform to foster the exchange reproducibility, and reuse of results.

3.5.4. Assessment of Machine Learning Frameworks

Computer science has been studying the field of machine learning for many decades. This extensive study has led to a vast variety of different methods, approaches, technologies, and implementations [27]. Especially for software engineering projects and implementations, the reuse of components and libraries is essential. In order to implement the machine learning functionality, as stated in Section 3.4.1, we assessed different freely available frameworks. A few basic considerations were driving our assessment:

1. The framework shall natively support machine learning for NLP tasks, such as classification, and text representation.
2. The framework should be able to support active machine learning.
3. The framework should be freely available, e.g., open-source.
4. The framework should be maintained and supported by an active community.

Following these basic considerations, Muhr (2017, pp. 46) conducted an extensive study of existing machine learning frameworks. Based on this assessment, we were able to draw the final decision to re-use the existing framework of Apache Spark and the MLLib.

[26] https://www.cs.waikato.ac.nz/ml/weka/, accessed on September 3, 2018

[27] A collection of different implementations are collected here: https://github.com/josephmisiti/awesome-machine-learning, accessed on September 3, 2018

3.5.4.1. Framework Overview

MLLib: The Machine Learning Library, MLLib, is a component on top of the Apache Spark ecosystem, which is a "fast and general engine for large-scale data processing"[28]. MLLib extends Apache Spark with machine learning functionality, which is currently available in version 2.2[29]. It supports the data handling capabilities, iterative batching and continuous streaming, of Spark, and is capable of handling large datasets.

Within the MLlib library, a variety of efficient and scalable implementations of common ML technologies are seamlessly implemented, such as standard learning algorithms for classification (e.g., naive bayes, support vector machine, neural networks, logistic regression) or clustering (e.g., k-means). Furthermore, the library offers basic methods to transform the textual input data into representations that are suitable for text classification. These include bag of words representation and feature set methods, such as TF-IDF.

A benchmark test in 2014 has proven the efficiency of Spark and MLlib. Assigned with the task of sorting a large dataset (100TB) Spark was three times faster using ten times fewer machines than the Hadoop MapReduce implementation. The sorting was performed on HDFS and an average sort rate of 7.27 TB/sec was reached in average[30]. Considering the execution of algorithms, Spark MLlib shows excellent performance and is continuously under improvement. Meng et al. (2016) compared the machine-learning frameworks Apache Mahout, using Hadoop MapReduce, and MLlib. They could show, that MLLib has advantages in efficiency of resources and runtime.

The fast evolution of Apache Spark can be explained by the large number of contributors and the support of an active community. Since 2009, more than 1,000 developers have contributed to the ecosystem. Apache Spark and its functionalities are well-documented. This includes code examples and details for the implementation of certain algorithms.

Mahout: Apache Mahout is an open-source project by the Apache software foundation. It was originally designed to support MapReduce with a focus on recommender engines, clustering, and classification (see Owen et al. 2011).

With the release of version 0.10 in April 2015, the Mahout implementation shifted from the focus to MapReduce, since the latest benchmarks have already shown better performance measures for other data-processing concepts and implementations (see description about Apache Spark and the MLLib). The focus has been set on a math environment called Samsara, which provides statistical operations, linear algebra, and data structures. The objective of Mahout-Samsara is to provide an extensible programming environment for Scala to enable end-users to develop their own distributed algorithms, instead of providing a machine-learning library with existing algorithms.

Currently, Mahout is available in version 0.13, released in April 2017. The latest documentation shows how the concept of Mahout has away from being a machine-learning

[28] https://spark.apache.org/, accessed on September 3, 2018
[29] https://spark.apache.org/docs/latest/ml-guide.html, accessed on September 3, 2018
[30] https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html, accessed on September 3, 2018

library for classification to being an "add-on" for other processing engines. The three basic classification algorithms of the original library[31], namely logistic regression, naive bayes, and hidden markov models, are deprecated. Currently, the system consists of an algebraic backend-independent optimizer and a Scala domain-specific language (DSL) consolidating the distributed and in-memory algebraic operators.

Hence, Mahout is an ML engine that requires deeper knowledge of both mathematical and programming skills. Moreover, due to the many dependencies, the initial configuration of Mahout may be difficult. The sparse documentation makes this even more challenging (see also Muhr 2017, pp. 51).

Compared to Spark, and thus also to MLlib, community support is low with only 24 contributors.

Weka: The "Waikato Environment for Knowledge Analysis" is a general-purpose open-source software workbench incorporating a variety of ML technologies. It was originally developed as an internal project written in C at the university of Waikato in New Zealand and was made publicly available in 1996 (see Holmes et al. 1994). In 1999, it was completely rewritten in Java. Today, it is available as version 3.8 under the GNU General Public License[32].

Weka provides a comprehensive and mature collection of data-preprocessing tools and ML algorithms (see Hall et al. 2009). One reason for the huge success is the graphical user interface (e.g., explorer and workbench) that enables end-users to access ML functionality; even if someone is not particularly familiar with programming. The UI allows for basic descriptive and statistical analysis.

Weka is a large collection of machine learning libraries and offers an open Java interface to foster the reuse in other applications. For the processing of textual information Weka provides implementations of common preprocessing steps (e.g., stemming, stop-word removal), as well as unsupervised and supervised ML algorithms, such as naive bayes, perceptron or support vector machines. For the evaluation of the classifier's performance, common measures such as precision, recall and F1 score, and the Receiver-Operator-Curves are available.

Originally, Weka was not designed for text classification tasks, but it can process textual data [33]. The textual data must be pre-processed using available components, such as the "StringToWordVector". These convert the text into a word vector. Based on this vector representation, further processing is possible. For example, using common TF-IDF methods and inputing the resulting features into existing classification algorithms.

A benchmark examining the efficiency on classification tasks is not available. For larger datasets, the efficiency decreases as the data has to be retrieved iteratively from external storages. These shortcomings are addressed by wrappers allowing for the continuous

[31] http://mahout.apache.org/users/basics/algorithms.html, accessed on September 3, 2018
[32] https://www.cs.waikato.ac.nz/ml/weka/, accessed on September 3, 2018
[33] https://weka.wikispaces.com/Text+categorization+with+WEKA, accessed on September 3, 2018

streaming of data. Weka provides extensive documentation about ML in general, and about the implementations of these algorithms.

The Weka toolkit has already successfully been used in the context of legal text classification (see Grabmair et al. 2015; Maat et al. 2010; Goncalves and Quaresma 2005) and in an active machine learning environment by Cardellino et al. 2015.

Scikit-learn: The scikit-learn framework (see Pedregosa et al. 2011) is an open-source general-purpose ML library for the Python programming language. The most current version is 0.19.0[34] under the permission of a Berkeley Software Distribution (BSD) license. The project started in 2007 with the aim to make ML available to non-experts by providing a reusable tool. This should foster its re-use in scientific and industrial contexts. Scikit-learn is purely written in Python and does not provide a native support for Java.

Scikit-learn supports several options for data mining and machine learning. It has a large variety of supervised (and unsupervised) algorithms for classification (e.g., support vector machine, naive bayes), regression algorithms (e.g., logistic regression) and clustering (e.g., kNN). All scikit-learn classifiers support multiclass classification. This is implemented as a one vs. all strategy (see Bishop 2006). To handle large datasets, an incremental loading and training strategy has been implemented: only a small number of instances is loaded into the main memory at one time and given to the classifier. This smaller batches are easier to handle and require less memory.

Scikit-learn offers a several components for the analysis and processing of textual data. Text is transformed into a feature vector (e.g., bag of words representation). Besides simple word vectors, Feature Selection (FS) techniques, such as TF-IDF methods, are supported. Several common components to perform basic preprocessing of text data, such as stop-words removal and stemming are provided by the framework. Scikit-learn has an integrated pipeline concept, in which different subsequent processing phases (e.g., conversion into feature vector, preprocessing, classification) can be combined.

In order to assess and evaluate the performance of algorithms and classifiers scikit-learn integrates all common classification evaluation metrics such as accuracy, F1, precision and recall, and Area Under the Curve (AUC).

Scikit-learn comes along with a detailed documentation covering the important parts of the API, including code examples, explanations, and tutorials. Additionally, it describes internal processes and architecture in detail, as well as common ML topics and algorithms. With more than 100 contributors since 2010, scikit-learn has a remarkable and very active community. The framework is also used for commercial purposes, e.g., Spotify.

Mallet: The MAchine Learning for LanguagE Toolkit (Mallet)[35] "is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text" McCallum (2002).

[34] http://scikit-learn.org/stable/, accessed on September 3, 2018
[35] http://mallet.cs.umass.edu/, accessed on September 3, 2018

It was initially released in 2002 and is released under Common Public License[36]. Compared to the frameworks mentioned before, Mallet is a small project.

Unlike the other general-purpose ML frameworks, Mallet is specifically designed for NLP, particularly focusing on document classification, sequence tagging, topic modeling, and numerical optimization. Mallet is written in Java and an easy-to-access API for external Java applications is provided. Like the other frameworks, Mallet offers routines to transform text documents into numerical representations (feature vectors), which serve as input for the classifier implementations. Similar to Spark, Mallet offers a pipeline model to configure preprocessing steps, for example the conversion into lowercase letters or the removal of stop-words.

With respect to classification algorithms, Mallet does not have that large range of options compared to other frameworks. Of the classifiers described in detail in the previous chapters, only NB is offered. Multi-label classification is not supported. However, for the evaluation of classifiers, common measures like accuracy, precision and recall, and the F1 are available. In contrast to Spark or Mahout, and similarly to Weka, Mallet is generally not aligned to import or process extremely large amounts of data.

On the website, one can find tutorial presentations about the concept Mallet implements. Additionally, there is a brief developer guide demonstrating the classification process, including some code snippets. Analyzing the code repository shows that the Mallet community (developers and users) is active ,but small compared to Apache Spark. In 2017[37], 27 active contributors could be counted on GitHub[38].

3.5.4.2. Conclusion

Our investigations and analyses, summarized in Table 3.6, show a compact comparison of different available and mature frameworks in ML. We have chosen to use Apache Spark and its MLLib for the machine learning component in our text analysis engine. Although there are emerging frameworks with a particular focus on deep learning, which becomes increasingly relevant in the field of image recognition or NLP, Apache Spark fully reflects a modern and state-of-the-art ML framework. This statement can be backed up with observations, that can be based on analyzing recently published scientific articles: according to Google scholar, 326 different articles have cited the main publication of Meng et al. (2016) since its appearance. This indicates the dynamics and relevance of Apache Spark for data mining and knowledge discovery tasks. Table 3.6 provides an additional overview and comparison referring to the high-level requirements stated in Section 3.5.4, namely support of NLP tasks, active machine learning, open-source availability, and community support.

The table lists the different frameworks (columns) and the evaluation criteria (rows). Each cell contains the result of the degree of support and fulfillment. It shows, that the Apache Spark with its MLLib has – based on our criteria and objectives and together with scikit-learn

[36] http://mallet.cs.umass.edu/about.php, accessed on September 3, 2018
[37] accessed on September 3, 2018
[38] https://github.com/mimno/Mallet, accessed on September 3, 2018

– the highest degree of fulfillment. Due to the fact that the remaining framework LEXIA is already implemented in Java, we have chosen Apache Spark as the framework in which the ML functionality is going to be implemented. This would have not been necessary, since the integration of the ML functionality is done via a service-oriented approach in which the NLP and ML are consumed on a well-defined interface. However, remaining in the Java ecosystem seems reasonable for maintenance and there was no main reason against this decision. The service-oriented approach, which is extensively described in Section 3.5.5.3, also allows to consume other frameworks or to easily change the machine learning framework used. Again the modularity in terms of re-usable software components is ensured. Currently, different, highly specialized frameworks for machine learning are developed and created. As it is unclear, in which direction this trend will evolve, the service-oriented architecture seems reasonable for this technological innovation. Additionally, separating the ML functionality in an individual service allows to scale the infrastructure more easily. ML is known to be very resource intensive, therefore having a service on a separate (physical) machine, e.g., server, facilitates independent scaling and faster NLP and classification.

Section 3.5.5 introduces the active machine learning process in details It unveils the process and the rationale, clarifies basic terms and their meaning, discusses its role in legal text classification and the architectural integration into the overall semantic analysis framework.

Criterion	MLLib	Mahout	Weka	Scikit-learn	Mallet
Version	2.2	0.12.2	3.8	0.19	2.0.8
License	ASF	AFS	GPL	BSD	CPL
Processing Platform	Apache Spark, MapReduce	Apache Spark	Wrapper for Spark	none	none
Interface Language	Java, Scala, Python, R	Scala, Java	Java, R	Python	Java
Large Dataset support	good	good	moderate	good	moderate
Community	good	moderate	good	good	moderate
Documentation	good	moderate	good	good	moderate
Text analytics and NLP support	good	moderate	good	good	good
Algorithms for classification					
Naive bayes	✓	✓	✓	✓	✓
Support vector machine	✓	only via MLLib	✓	✓	×
Multi-Layer perceptron	✓	only via MLLib	✓	✓	×
Multiclass classification	one vs. all	only via MLLib	one vs. all and one vs. one	one vs. all and one vs. one	×
Multilabel classification	one vs. all	×	×	one vs. all	×
Pipeline configuration	✓	×	×	✓	×
Preprocessing					
Stemming	✓	×	✓	✓	✓
Stop word removal	✓	×	✓	✓	✓
FS methods	good	moderate	good	good	moderate
Text representation					
Word vector	✓	✓	✓	✓	✓
TF-IDF	✓	×	✓	✓	×
Evaluation					
Confusion matrix and derived measures	✓	✓	✓	✓	✓
Receiver operation char. and area under curve	✓	×	✓	✓	✓

Table 3.6.: Comparison of different frameworks regarding ML and software engineering criteria (extension of Muhr 2017).

3.5.5. Active Machine Learning

Based on the comparison of potential frameworks, a decision for the ML component has been made. We have chosen to implement the ML functionality using Apache Spark. This section will provide a more detailed introduction into Active Machine Learning (AML) and describes the idea and basic terms (see Section 3.5.5.1). In addition, the role of AML within the classification of legal texts is discussed in Section 3.5.5.2. Based on these considerations, the questions remains how this functionality can be integrated seamlessly into the overall process of semantically analyzing and processing textual data. The conceptual integration and the interaction with the implemented prototype are discussed in Section 3.5.5.3.

AML is a specific form of semi-supervised machine learning (see Settles 2010). It relies on the integration of a domain expert, i.e., a human person, during the learning process. Recent studies have shown, that this integration could significantly improve the learning behavior of the ML classifier (Olsson 2009). Due to its nature, it is also called an "optimal experimental design"[39].

The next section will briefly explain the general idea behind AML by providing an overview of the overall process and introducing the basic concepts. In addition, we will describe its role within the domain of legal text and legal entity classification. Finally, we will discuss the technological concept in detail and its architectural integration into LEXIA.

3.5.5.1. Active Machine Learning: The Process and Foundations

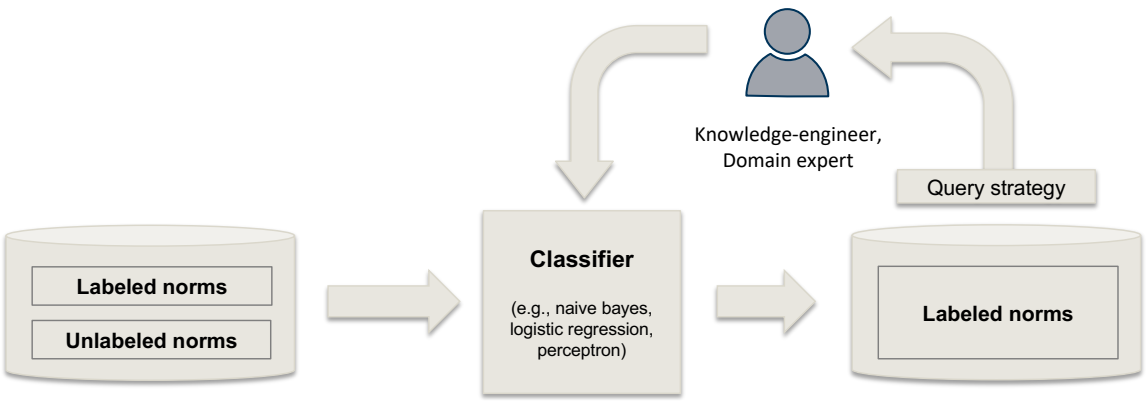


Figure 3.19.: AML process for the classification of norms describing the interaction between classifier, strategy, and domain expert (see Waltl et al. 2017b).

A general overview of AML is displayed in Figure 3.19. The process starts by providing two different pools of instances. The figure already shows the adaption for the classification of legal content, namely of norms. An existing dataset, e.g., a set of norms, exists which can be divided into two categories: labeled and unlabeled. This dataset serves as the input for a machine learning classifier.

[39] https://en.wikipedia.org/wiki/Active_learning_(machine_learning), accessed on September 3, 2018

This classifier is a mathematical algorithm that learns how to recognize and predict unlabeled instances based on the input data. The input data is mapped into a set of so-called features. These features can be parsed by the classifier and stored within an efficient internal state, a so-called model. A very common and simple feature representation, that is widely used in NLP is the bag-of-words model. Thereby, the textual input data is separated and split into its words. These words are stored within a multiset that does neither preserve the order nor information about its grammar. It only keeps the number of occurrences for each word. This multiset is one feature processed by the classifier and used to train and predict the labels of the instances.

For AML, a large variety of different classifiers can be used. In our implementation, we have used three main classifiers, namely

- naive bayes,
- logistic regression, and
- a perceptron.

This selection covers main representatives from different learning strategies, namely discriminative vs. generative (see Ng and Jordan 2002). Naive bayes is known to be a famous representative for discriminative learning, whereas logistics regression belongs to the category of generative learning classifiers. The last candidate in our list, the perceptron, was selected from the list of neural networks. A more detailed description of machine learning classifiers has been provided by Russell and Norvig (2009). Modern software frameworks for machine learning offer a large variety of different classifiers, that can easily be re-used. In our implementation, we have used the Apache Spark[40] framework for machine learning. A detailed comparison of frameworks with their advantages, disadvantages, and limitations was carried out by Muhr (2017).

Once the classifier has been selected and trained the instances, e.g., norms, are labeled. The classifier assigns a label to each instance. This is done regardless of any confidence level or threshold. Based on these labeled instances, a query strategy selects a pre-defined number of instances from the pool of labeled data. The query strategy follows an mathematical principle and is designed to select those candidates which are most beneficial to the classification if they are labeled correctly. Different strategies exist for this purpose, which are summarized in Table 3.7.

Table 3.7 shows only a selection of the query strategies and new strategies are constantly being proposed. Currently, the field of AML is very vital and intensively studied. However, we have made experiments to show the applicability of these query strategies to classify legal entities (see Section 6).

Based on the batch size, which specifies how many instances are chosen by a query strategy, a domain expert is involved into the learning round. The selected instances are shown to the domain expert who needs to provide the labels for the instances that are shown he is shown. This active integration of external knowledge, like an oracle, is the essence of AML. The classifier is no longer thrown back to the training data it receives, but can "ask" a domain expert to efficiently train and improve the classification process. Once the domain expert has finished

[40] https://spark.apache.org/, accessed on September 3, 2018

Query Strategy	Method	Description
Uncertainty Sampling (US)	Entropy	Selection based on the average information content (Shannon entropy) of an instance.
	Margin Sampling (MS)	Selection based on the output margin of the predicted outcomes with the highest probability.
Query by Committee (QBC)	QBC Vote Entropy (VE)	Selection based on a committee of different QS methods (ensemble with majority vote).
	QBC Soft VE	Selection based on a committee of different QS methods (ensemble with majority vote, including probabilities).

Table 3.7.: Query strategies for active machine learning (see Waltl et al. 2017b; Muhr 2017).

his labeling this set is incorporated by the classifier which then refines his decision model and structure.

Using the updated classification model, the instances are re-classified and a new learning round is performed. This iterative process is carried out until either all instances are labeled or another stopping criterion is met. The performance of the scenario can be measured using conventional measures for machine learning, such as precision, recall, Receiver Operator Characteristics (ROC).

In the domain of legal text analytics, this approach can be used to decrease the number of labeled instances in order to perform the classification of legal entities. The main concepts of AML are summarized in Table 3.8.

3.5.5.2. Active Machine Learning: Legal Text Classification

According to Russell and Norvig (2009), machine learning has been used for text classification for decades. Common approaches addressed the analysis of the sentiment within a given document or the categorization of emails into two categories: spam or not-spam. Thereby, the usage of simple representations of the documents are usually bag-of-words. One drawback of this representation is, that the feature vector is very large and sparse. For example, if the overall vocabulary consists of 1,000 different words, the vector for an email with 20 different words consists of 1,000 entries with 980 zeros and 20 ones. Good spam detectors use more features than just the unigrams, such as time of the message, sender's address. Russell and Norvig (2009, p. 866) conclude: "The choice of features is the most important part of creating a good spam detector—more important than the choice of algorithm for processing the features."

Based on this, it seems to be straight forward to reuse machine learning to differentiate not only between spam and not-spam, but to train algorithms in supporting information extraction tasks in legal texts. This has also been observed by Ashley (2017, p. 235): "Applying ML to legal texts will play key roles in a legal app for cognitive computing. One goal in cognitive computing is for ML algorithms to learn to identify patterns of textual features that are important for human problem-solving."

Term	Description
Scenario	Describes how the instances are queried and labeled by the user. This could be done sequentially (stream-based) or as a bulk of instances (pool-based).
Learning round	Describes a full iteration of an AML cycle, covering the labeling of instances by the classifier, querying instances by a given strategy, labeling of instances by a domain expert, and incorporating these into the training data set.
Classifier	Algorithms that learn from a given set of labeled data (i.e., training data) and apply this to unlabeled data (test) and predict their missing label.
Feature	Measurable property or characteristic of an observed phenomenon, such as words from the vocabulary, or other information from a document.
Batch size	Describes the number of instances that are queried within one learning round.
Seed set	Describes the number of instances that are queried within the first learning round.
Query framework and strategy	Mathematically optimized functions determining how instances are selected from the labeled pool of instances. A common method is uncertainty sampling, which is based on an entropy measure.
Stopping criterion	Defines the conditions to start another learning round or to stop the learning. By avoiding unnecessary learning rounds, the effort of AML can be kept low.
Performance measure	Learning curves based on the number of labeled instances, as well as on traditional measures, e.g., precision and recall, are applied to analyze the overall performance.

Table 3.8.: Basic terms and their descriptions in the context of AML (based on Muhr 2017).

Beside the challenge of extracting the features and setting up the technological platform enabling this form of advanced legal data analytics the question remains what type of human problem-solving should be supported. In Section 4, we will discuss the analysis of statutory texts in Germany. Starting from the analogous process, which has been well-studied in legal theory, we will derive and formalize a legal interpretation process connecting the formalization and software-supported analysis of legal documents.

As described in Muhr (2017), we have performed a small proof-of-concept to automatically categorize legal documents into three different categories: laws, judgments, and miscellaneous. We used an annotated corpus of 132,000 documents from the domain of German tax law[41] and randomly selected 1,000 documents to perform the experiment. To efficiently handle large document corpora we selected the first 4,096 characters of each document, splitted the string into tokens, removed stop-words, and represented them in a bag-of-words model. Within this proof-

[41]The corpus was created and annotated by an industry partner of the research group.

of-concept, we used three different classifiers (naive bayes, logistic regression, and a perceptron). We showed that the classification task can be performed very well: $F_1 = 0.96$. Furthermore, we showed that using AML significantly improves the learning curve of the classifiers. We set up the experiment in such a way that the initial seed set covered 15 documents and in each learning round, we provided the labels of another 15 documents. Overall, this resulted in 97 learning rounds. We measured the performance of the classifiers after each round and after 2-3 rounds, the AML set-up has always been superior to the classical ML, which used random query strategy to train the classifier. The experiment shows the potential of AML in the domain of document classification. As we will see in Section 4.1, the classification of documents is not required during the analysis or interpretation of a legal text, but the classification of legal norms and their semantic role within a statutory text is.

Savelka et al. (2015) have also described the experiment for a classification task. They have used a binary classification task to train an AML classifier to determine (labels: relevant, not-relevant) whether a provision is relevant or not with respect to a legal issue and given statutes. They discuss the potential of their modest results as stated by Savelka et al. (2015, p. 10): "[...] automating the process [of relevancy assessment] still seems quite distant."

In Section 6.4, we will discuss the application of using AML to support the analysis of German texts, by automatically classifying sentences statutory texts into functional categories, as they are used during the analysis, interpretation, and application of legal norms.

3.5.5.3. Active Machine Learning: Concept and Architectural Integration

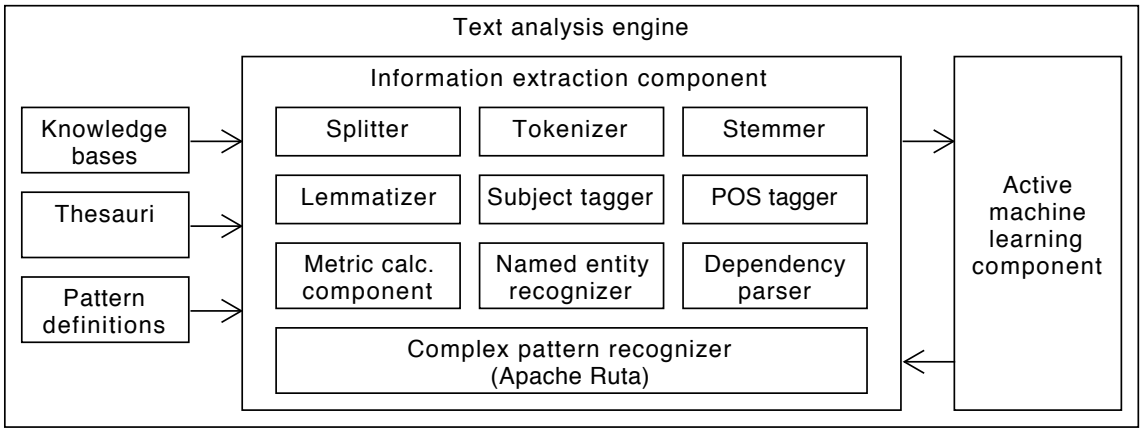


Figure 3.20.: Black box overview of the AML component and its interaction with the information extraction component.

As shown in Figure 3.20, the AML component is logically a part of the text analysis engine, but implemented as a separate component. This component was realized as a separate service that can be consumed via an interface. This is mainly due to the fact that the training and predicting phases are resource-intensive tasks. This mainly concerns processing time and memory. To

enable scaling to larger datasets, we have decided to implement this component in a way that it can also run on another (distributed) infrastructure, e.g., physical server, and virtual machines.

Another benefit is that the modularity of the system is preserved. Once the AML components changes, e.g., for the reason of technological innovation, the impact on the text analysis engine system is minimal. Since the information extraction component consumes it, like every other component as a black box. It does not depend on the internal structure or internal components.

The information extraction component exchanges data with the AML component via well-defined Representational State Transfer (REST) interfaces. Thereby, the data flow mainly consists of datasets (e.g., instances) towards the AML component and prediction results backwards. In addition to the datasets, some information about the learning configuration, e.g., seed size, batch size, learning rounds, are exchanged.

Figure 3.21 visualizes the AML component as white box view. This allows a more comprehensive inspection of the implemented components and how they are interacting with each other.

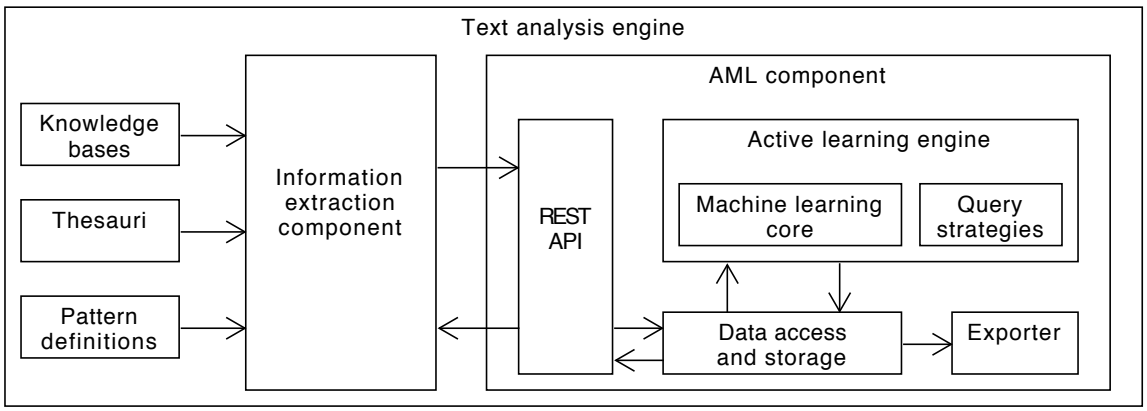


Figure 3.21.: White box view of the AML component and its integration into the text analysis engine.

The AML was implemented as a stand-alone server component, which can be hosted on any (distributed) infrastructure. It is again a web application and was developed using the Play Framework[42]. The component consists of four, respectively six different components, which are briefly described:

REST API: The API handles the data exchange between the information extraction component and the AML component. It maps the data objects into JSON objects which are exchanged using normal HTTP requests. The data flow mainly consists of the exchange of instances from the information extraction towards the AML component. Training and test data sets are received thereby. The API passes the results of an executed classification task, i.e., prediction, back to the information extraction component.

The parametrization of a concrete classification task can is also be done via the API.

[42]https://www.playframework.com/, accessed on September 3, 2018

For the classification task, necessary information, such as the seed size, batch size, query strategy, classifier instance, and the stopping criteria, are also exchanged.

Data storage: To store data locally, the AML component contains a separate data storage component. This storage keeps the local configurations of trained models as well as training and test data sets. For our implementation, a MongoDB[43] was used. Storing data locally decreases the amount of data that needs to be exchanged between the information extraction and the AML component.

Exporter: The exporter component logs information along the learning procedure. This information is stored in Comma-separated values (CSV) files and can be analyzed and evaluated to draw conclusions about the classifier performance. Depending on its parametrization, it logs each learning round or the final result of the classifier.

Active learning engine: Whereas the REST API, the data storage, and the exporter are components to set up a proper infrastructure, the concrete learning process, training of the classifier, and predicting potential outcomes is performed in the active learning engine. It consists of two main components: the machine learning framework and the query strategies.

Machine learning core: In modern software implementations, it is no longer necessary to start from scratch and to re-implement classifiers, such as naive bayes, logistic regression, or a perceptron. A large variety of software frameworks exists, providing very efficient implementation and ready-to-use APIs. These decrease the effort of using (active) machine learning within an application.

We have analyzed different machine learning frameworks and assessed their suitability for reuse within our environment (see Sections 3.5). Finally, we have decided to use and adapt Apache Spark and the MLLib[44] package.

Apache Spark is implemented within a Java web application environment, is a general-purpose machine learning framework, and supports AML for text classification. The preprocessing of the documents, e.g., importing, normalizing, segmenting, and the feature selection is done by the information extraction component.

Query strategies: Beside the machine learning frameworks and basic implementation for handling of data and instances, we provided several query strategies to compare these with each other. The query strategies have already been discussed in Table 3.7. The strategies can be used independently from the underlying classifier. They mathematically describe how the instances to be labeled by a domain expert are selected. Based on formal specifications and measurements, such as entropy and information gain, the most informative instances are selected.

Decoupling the query strategies from the remaining machine learning components fosters the modularity of the overall system and allows an easy integration of new strategies or classifiers. The remaining system and workflow is hardly effected.

[43] https://www.mongodb.com/de, accessed on September 3, 2018
[44] Version 2.1.1. as of May 2, 2017

3.5.6. Apache Spark

Apache Spark[45] is shipped with a variety of libraries that fit together seamlessly. The main components are shown in Figure 3.22.

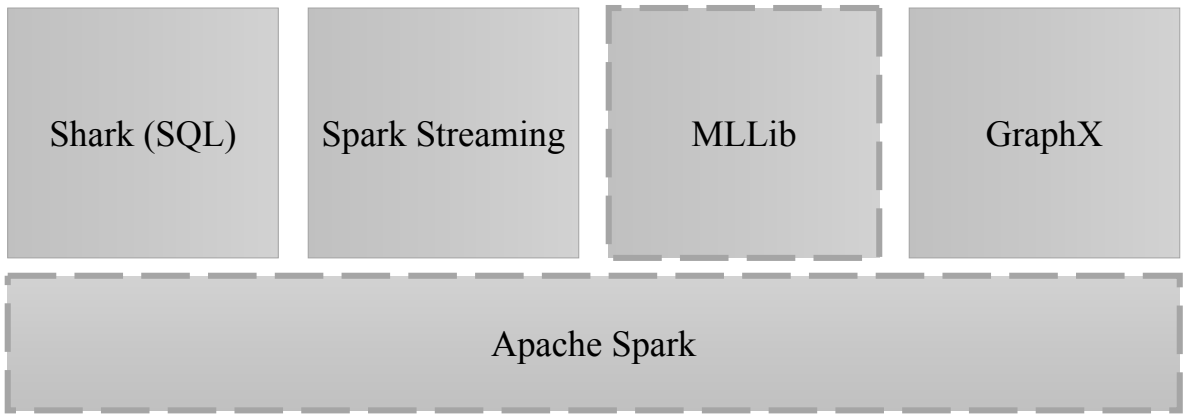


Figure 3.22.: Apache Spark stack including libraries for efficient data management (i.e., Shark and Spark Streaming), machine learning (i.e., MLLib), and processing of graphs (i.e., Graph).

As we are only interested in the ML components in our implementation we have adapted and configured the MLLib for the classification of legal entities and used the core functionality of Apache Spark for the preparation of data (i.e., DataFrames) and for the data exchange with MLLib.

3.5.6.1. MLLib: Machine Learning with Apache Spark

As already stated above, the MLLib contains a large variety of functions and machine learning components. For our purpose, namely the classification of text, the following methods are implemented[46]:

Binary classification	**Multiclass classification**	**Regression**
linear SVM	logistic regression	linear least squares
logistic regression	decision trees	Lasso
decision trees	random forests	ridge regression
random forests	naive bayes	decision trees
gradient-boosted trees	multilayer perceptron	random forests
naive bayes		gradient-boosted trees
multilayer perceptron		isotonic regression

Table 3.9.: Overview of main classifiers supported by MLLib.

[45] http://spark.apache.org/, accessed on September 3, 2018
[46] MLLib version 1.3.0.

Table 3.9 lists the different classifiers that are implemented in MLLib and differentiates them according to their problem type: binary classifications, multiclass classification, and regression.

In our use case, only the classifiers supporting multiclass classification are relevant. Since binary classifiers split a given instance into dichotomous classes ('0' or '1'), multiclass classification can have multiple classes (i.e., semantic types) to which an instance could potentially belong. We have selected

1. logistic regression,
2. naive bayes, and
3. multilayer perceptron

as classifiers for our active machine learning component. Logistic regression and naive bayes are common classifiers for text and document classification, and the multilayer perceptron is a simple form of a neural network, and we wanted to compare them. In addition, we cover the

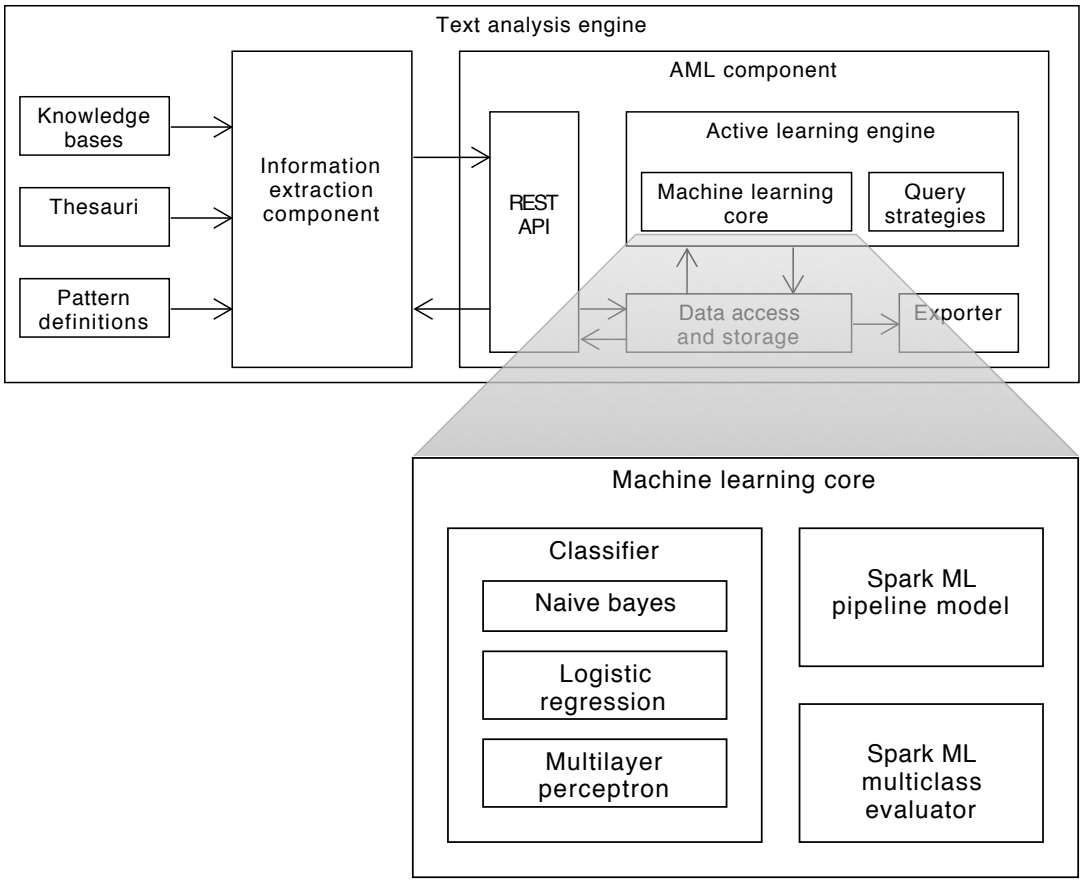


Figure 3.23.: The machine learning core, part of the active learning engine, based on MLLib.

main learning strategies for machine learning classifiers, namely discriminative learning (e.g., naive bayes), and generative learning (e.g., logistic regression).

3.5.6.2. MLLib: The Machine Learning Core

The detailed integration of MLLib is shown in Figure 3.23. It visualizes the AML component and its integration into the text analysis engine. The components REST API, data storage, exporter, and query strategies have already been described in Section 3.5.5.

The machine learning core is based on MLLib and mainly consists of the mentioned set of classifiers, the Spark pipeline model, which allows the configuration of components within pipelines, and the Spark ML multiclass evaluator, which allows a detailed assessment of the performance within individual learning rounds.

Although this fine granular encapsulation increases the effort during the planning and conceptualization phase, it however fosters the reusability and modularity of the overall system and allows for an easy integration of new components. Although we have made an extensive study of available frameworks, it cannot be excluded — in fact: it is very likely — that new components or libraries will be implemented that provide even better functionality and performance. The overall system with its architecture and components could remain, whereas these new libraries can easily be integrated.

3.6. Summary

This chapter described the concept of semantic analysis in general and its applicability within the legal domain. It illustrated different notions and concepts for the software-supported linguistic analysis of legal documents and provided the conceptual background for an interpretation support for statutory texts.

A reference process has been developed that discusses the interaction between activities, roles, and tools along three different activities: import, analysis, and application. The reference process shows how text analytics can be integrated in this interdisciplinary field. The usage of software is to support and assist during workflows and to meet the information needs of legal practitioners and scientists more efficiently. The chapter discussed the importance of linguistic models and the role of annotations and deepens the understanding of interaction between humans and software in the challenging task of semantic analysis.

Based on the processes, this chapter clarified the meaning of semantic entities and annotation types. Using a modeling perspective, it is discussed if and how it can be possible to assign a semantic type to a region of text, i.e., annotation. Starting from basic, linguistic, and named entities a more advanced notion of legal entities is introduced and discussed.

At this stage, the base line is set for the discussion how technology is now able to represent this process and to provide concrete functionality achieving this overarching objective. It begins with exemplifying the annotation of legal documents, which could either be done manually

or automatically, and how these annotations are assigned to legal documents and properly persisted.

A system that manages large document collections and manually and automatically created annotations was built upon the prior considerations. The system's architecture has the capability to import, store, analyze, and visualize documents and their annotations. In addition, it contains an advanced and flexible text analysis engine fostering the reuse of components from computational linguistics. An established baseline architecture has been used to implement the system: Apache UIMA and Pipes & Filters architecture.

An additional ML component has been implemented that offers possibilities for supervised machine learning in order to constantly refine and improve the possibility of extracting legal entities. Therefore, an open-source framework, namely Apache Spark, was seamlessly integrated. Using so-called AML the classifiers are trained efficiently and enrich the text analysis engine.

Based on this conceptual consideration and the system, the next section is going to discuss the interpretation of statutory texts in Germany. A reference process, specifying the analysis and interpretation of statutory text, is introduced for this purpose. It discusses how legal norms can be interpreted and how they can be transformed into model-based decision structures. The process also elaborates on the role of software-supported analysis of legal documents.

CHAPTER 4

Concept and Design of a Model-based Reasoning Framework

This chapter describes the formalization of statutory texts from the German legal domain, i.e., laws. Thereby, the idea of model-based reasoning for legal texts is conceptualized and a logical calculus is developed that enables end-users to capture the interpretation results as a formal representation: a model. The process of analyzing and interpreting of German laws is studied in detail, and developed on commonly accepted legal theory in Section 4.1.

Based on these considerations, the model-based reasoning is described in depth with a particular focus on the provision of complementary representations, namely statically and dynamically (see Section 4.2). It extends prior approaches of formalizing decision structures into legal ontologies by introducing the Model-based Expression Language (MxL), a high-order expression language, similar to description logic, allowing for reasoning on entities, relationships, and attributes. MxL was particularly designed to support the end-user-enabled analysis of complex linked data as introduced by Reschenhofer (2017).

On the foundation of the concept and the calculus, the role of software-supported semantic analysis of legal documents during the interpretation and formalization is elaborated in Section 4.3. Thereby, user interaction, form-based usage, and execution of model-based decision structures are evaluated. Finally, the potentials of analysis, inspection, and explanation of the formalized models are shown in Section 4.4.

4.1. Reference Process to Formalize Statutory Texts

The formalization of statutes, e.g., laws, is known to be complex and challenging. A large variety of complex concepts have been studied within the last decades. A good overview is provided in Bench-Capon et al. (2012). Recently, Ashley (2017) summarized the most active

and promising research areas within the field of artificial intelligence and law. It nicely shows how researchers from different jurisdictions, e.g., common law, and civil law, have developed different concepts addressing the particularities of their field of studies.

The main problem is the assurance of the semantical equivalence between a textual representation and a formalized representation. Whereas the latter can be executed by computer systems, i.e., software, the former is still the predominant form of how legal information is published, this holds from both sources of law: legislative and jurisdictional texts.

The problem of preserving the semantical equivalence is known as "isomorphism"[1]. The phenomenon and potential criteria to allow for isomorphism were extensively described by Bench-Capon and Coenen (1992). As of today, however, this issue has not sufficiently been solved and there does not seem to be a solution for the problem and two related sub-problems:

1. Transferring a text into an executable representation, and
2. proofing its equivalence.

In the field of legal informatics, the isomorphism was reduced to something that could be called "weak" isomorphism. Bench-Capon stated: "The important demand made by isomorphism is that there is a clear correspondence between items to be found in the source material and items to be found in the knowledge base." (Bench-Capon and Coenen, 1992, p. 67) Reading this quote, attention has to be drawn to the term "correspondence". Bench-Capon followed the idea of Karpf (1989), by reducing the isomorphism to a reference problem. Although this does not solve the original problem, it seems to be a constructive and implementable contribution. Within our research, we have adopted this idea of "weak" isomorphism and correspondence and applied it to the domain of the analysis and formalization of German legislative texts.

We thoroughly analyzed the potential of model-based reasoning to formalize decision structures that emerge from German statutes, i.e., laws. Based on legal theory, especially by Larenz and Canaris (1995) and Hart and Green (2012), the process of the interpretation of legal texts has been studied and formalized. We embedded the interpretation process int the software-supported semantic analysis of legal documents and formalized it based on the Rationale Unified Process by Kruchten (2004), which differentiates between the following elements (see also Section 3.1 for a discussion of the RUP):

Activities: An activity summarizes a unit of work that must be performed.

Roles: Individuals or groups performing activities of the process.

Services and Tool support: Identifies the role of software, by supporting involved roles in their activities, along the process.

Based on the RUP, we have developed an interdisciplinary process that integrates the activities of analyzing and interpreting legal documents. The process reflects different activities, that are performed subsequently or iteratively. This contributes to the domain knowledge required in order to develop appropriate software-support. Beside the software support, we identified the different stakeholders and roles in the process and the activities.

[1] from the Ancient Greek: isos "equal", and morphe "form" or "shape"

4.1.1. Reference Process

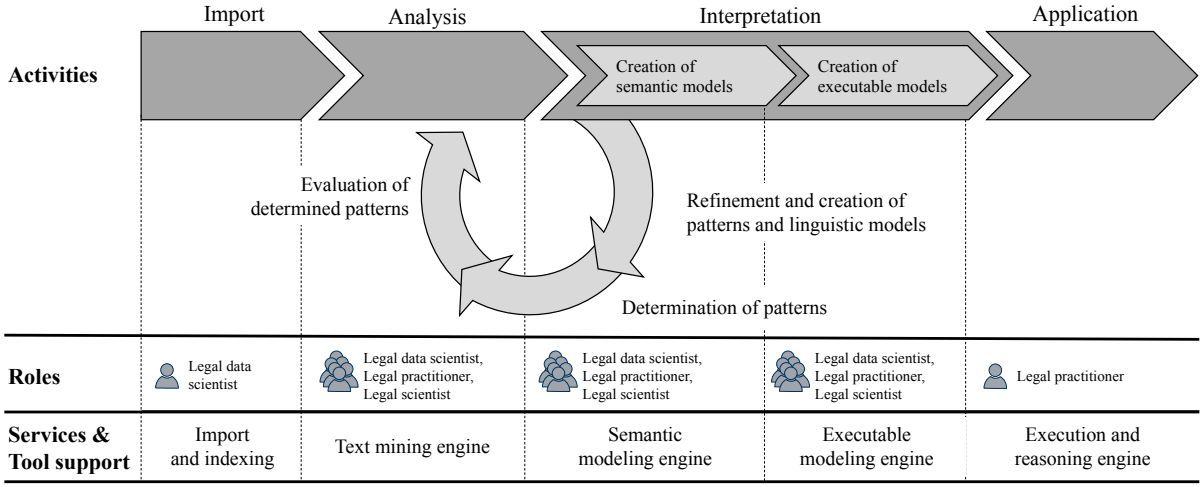


Figure 4.1.: Reference process for software-supported interpretation and formalization of legal documents (Waltl et al. 2017c).

Figure 4.1 visualizes the overall process. The process is structured into three different rows, namely "Activities", "Roles', and "Services & Tool support". The three rows are divided into three subsequent main activities, namely "Import", "Analysis", "Interpretation", and "Application". The activity "Interpretation" is divided into sub-phases, namely "Creation of Semantic Models" and "Creation of Executable Models". In addition, there are three phases that are iteratively performed which connect the "Interpretation" and "Analysis" activity.

4.1.2. Activities

Import: The process starts with an import activity. During this phase, required documents are imported and indexed into the tool. Although this phase seems to be straight forward, and the task is certainly not special in the domain of legal documents, the large variety of document types increases the effort that might have to be made. In addition, the required documents are not necessarily available in a digitized format suitable for processing. Challenges of digitization, such as OCR, can significantly increase the complexity and effort during this phase.

Analysis: During the analysis phase, the textual representation is enriched with semantic information. This information added as annotations. This could either be done manually, by a domain expert, or computer-supported (see Chapter 3). The rationale behind is to extract sentences and phrases that support the subsequent interpretation phase. The reference process does not specify the annotation types that need to be determined. This selection and taxonomy is highly domain-dependent. This difference in logic potentially leads to different semantic types. As we have already shown in Section 3.2.3, there is a huge difference between the semantic types for documents from US case law and laws from Europe.

Consequently, one can expect a difference between the semantic types for tax law and criminal law.

Subsequently, during the analysis phase, the manual and automated methods from NLP are applied to the legal text to support the interpretation process.

Interpretation: The interpretation phase describes the most important part of the reference process. This is when, based on the documents, the operationalization takes place. The text is analyzed by humans and put into the context in which it should be applied. This "putting" is essential. Based on the text, decisions are made and reasoning procedures are applied. The exemplification of these procedures and capturing their results is exactly what should be achieved within this phase. The overall goal is to make this process explicit, which is implicitly carried out by a legal practitioner or legal scientist.

This explication is supported by the differentiation into two different, but complementary representations: semantic and executable models (see also Section 4.2).

Creation of semantic models: During the interpretation phase, the knowledge required to fully represent a legal issue needs to be structured. Based on the text within a statute and documents required to interpret the text, such as articles, commentaries, the main factors are modeled within an ontology. The ontology captures types, attributes, and the relations between those types. The semantic mode formalizes the statute as it represents static concepts.

Creation of executable models: The executable model is an additional layer for the semantic model. Based on types, attributes, and the relations among types, the executable model formalizes the dependencies between these entity types. So-called "derived attributes" (see Section 4.2.6) are expressed as logical formulas, which can be automatically be evaluated. Derived attributes operate on the semantic model and extends them with additional functionality which allow for automated reasoning on legal interpretations of statutes.

Iterative analysis: The interpretation phase is constantly supported by results from the analysis phase. It refines the annotations documents at hand with additional annotations, or by removing unnecessary and wrong annotations.
The cycle consists of three feedback steps:

1. Refinement and creation of patterns and linguistic models
2. Determination of patterns
3. Evaluation of patterns

Application: Finally, the model integrating the executable semantics can be applied. Thereby, the model is instantiated. This instantiation is done by providing concrete, real-world facts. Based on the model's information about reasoning procedures, it can automatically derive new knowledge. One has to be careful to declare declare this new knowledge "a final solution", since it most probably requires more than a concrete value or decision between "yes" or "no" in complex cases. There are however many intermediate results with exactly this form. The decision-making process is supported, not automatized.

Using this model, the application phase also allows for what-if analyses to get instant feedback on the consequences of changes within the input parameters.

4.1.3. Roles

Throughout the reference process, different roles are required. These are responsible for the document import, analysis of documents, as well as for the interpretation and for the related the creation of the models.

Legal data scientist: Mostly, the import phases and the analysis of indexed documents require technological background. During the import activity, the pre-processing of documents can be required. Documents are not necessarily available in a digital format that is already well-suited to be processed and analyzed automatically. Legal data scientists adapt the imports and perform the pre-processing, so that imported documents are well-structured and their content is fully indexed.

Legal practitioner & legal scientist: The domain expertise, which is required to interpret a statue accordingly, is provided by legal practitioners and legal scientists. Based on the annotated text, they create semantic and executable models. To formalize the models and the executable parts of it accordingly, they are supported by the legal data scientist. They have to interpret the written text with the ultimate goal of its final automated application. They need to determine how the decision structures within a given statute is going to be applied.

4.1.4. Services & Tool-Support

Some of the key elements are the software tools and services that assist in automatically perform the semantic analysis, support during the modeling, and the execution of the model-based decision structure.

Importing and indexing software: This service was already described in depth in Section 3.1.5. The particular role has not changed in the context of the interpretation and formalization of legal documents: The ultimate goal is to import and normalize relevant documents of different formats and types, if required. The normalization ensures the applicability of the text analysis engine (see Section 3.4.1.4).

Text mining engine: The imported documents are annotated to provide additional information during the interpretation activity. The text mining engine as well as the linguistic models are trained to fully support legal scientists and practitioners. Ideally, the automatically extracted semantic types reflect the information needed by the legal experts.

Semantic Modeling Engine: The required capabilities to capture the semantic model of a statute are provided by the semantic modeling engine. The requirements are extensively discussed in Section 4.3. The semantic modeling engine allows users to graphically create types with attributes and relationships between them. Thereby, the graphical representation follows the classical view of legal ontologies as common among ontology editors. Internally, the

modeling engine stores the data efficiently and allows for subsequent manipulation, such as updates.

To foster the above mentioned correspondence between text and semantic entities, the model elements, i.e., types, attributes, and relations, are linked to the text. Consequently, textual changes can automatically be determined and users can be notified to check potential changes in the semantic model.

Executable Modeling Engine: In addition to the semantic modeling engine, the executable model engine enables end-users to create expressions and constraints and assign them to derived attributes. Thereby, the user is supported with an online code editor assisting with the creation of expressions. The executable modeling engine is integrated into the overall editor and interacts seamlessly with the semantic modeling engine in the user interface. The requirements for this engine are discussed in Section 4.3.

The executable modeling engine also provides immediate feedback on derived attributes as to whether they are well-formed or not. Reschenhofer (2017) has already shown that even for technical tasks and technology-savvy users, the formalization into an expression language is not trivial. In conclusion, enabling end-users to formalize decision structures always requires some training and proper tool support.

Execution and Reasoning Engine: Once the decision structure is formalized into a model-based representation, the execution and reasoning engine can reason on these structures. The model can be considered as a template, which can be applied to different concrete legal cases. Therefore, users have to provide a set of facts, which are inputed into automatically created forms. Based on this set of facts, the outcome is determined.

This engine does not only determine the overall outcome of all specified derived attributes, but also static views on the abstract syntax trees and data flows (see Section 4.4). This is a step towards the explanation of the decision structures and can support the analysis and inspection of decision structures.

4.2. Model-based Reasoning

4.2.1. Ontological Models and Limitations of Description Logics

Computational models of legal reasoning have been studied in the fields of artificial intelligence and law since its emergence (see Ashley 2017, Part I). Whereas, the first approaches were called Legal Expert Systems (LES), a more differentiated view exists today. Most modern approaches focus on reasoning in legal argumentation: extracting, structuring, analyzing, or deriving new knowledge on complex legal argument structures. This supports different use cases, such as finding supporting or counter-arguments, automatically making an argument, or even predicting the outcome of cases.

Especially in common-law jurisdictions, to mine arguments from cases and to provide a system that supports during the above-mentioned use cases are considered to be very helpful. In civil law systems, the statute is still the pre-dominant source of law (see discussion in Section 4.1). This

is why this approach tries to formalize the interpretation of legal norms, as they are written in statutes. The baseline are so-called ontologies, which are common in today's information systems.

The usage of ontologies within the domain of legal informatics for either knowledge representation or reasoning is widely accepted. Wyner (2008) defined an ontology as follows: "An ontology is an explicit, formal, and general specification of a conceptualization of the objects and structural relations between those object in a given domain.". According to this working definition, ontologies are created in a particular domain for a specific reason. Consequently, ontologies are designed to reflect a specific and well-defined domain. Ontologies serve different purposes: they can be purely used for knowledge engineering and structuring of terms (e.g., WordNet by Miller (1995), GermaNet by Hamp and Feldweg (1997)), or they are used in more elaborated scenarios to enrich objects and relations with executable semantics, allowing automated reasoning.

State-of-the-art for reasoning within ontologies are description logics Krötzsch et al. (2012). Krötzsch et al. (2012, p. 1) describe the purpose of description logics and their role within ontologies as follows: "[...] allows humans and computer systems to exchange DL [description logic] ontologies without ambiguity as to their meaning, and also makes it possible to use logical deduction to infer additional information from the facts stated explicitly in an ontology - an important feature that distinguishes DLs from other modeling languages such as UML.". As of today, there is no standard in description logics. Krötzsch et al. (2012, p. 1) offer a potential explanation: "This is one of the reasons why there is not just a single description logic: the best balance between expressivity of the language and complexity of reasoning depends on the intended application." (Krötzsch et al., 2012, p. 1) This can be applied to the field of artificial intelligence and law, in which ontologies have been used to structure knowledge, but also to enable reasoning. However, almost every approach relied on the W3C standard OWL (see Sartor et al. 2011a). This standard is capable of deductive reasoning, but it lacks of the possibility to express algorithmic semantics between objects and attributes. On the contrary, this is essential in various fields in the legal domain, e.g., in tax law. It would be possible to model a taxpayer and related objects that influence his tax duties, such as employment, salary, etc. But description logic is not capable of expressing the arithmetical expression and to evaluate the amount of taxes one has to pay. This is just not the purpose it was originally designed for. In addition, the evaluation of these terms dramatically increases the computational complexity of description logics (see Ohlbach and Köhler 1999).

In conclusion, ontologies are already well-studied, widely used and practicable within the field of artificial intelligence and law. However, current artificial intelligence approaches focus on the deductive inference and only partially on expressing arithmetical reasoning. Based on an illustrative example, the next sections will introduce the usage of ontological modeling to formalize interpreted statutory texts, including the support of arithmetic expressions.

4.2.2. Formalization of Child Benefit

Within the tax law, a large variety of norms and regulations exists that govern the tax duties or the rights of retrieving a benefit from the state. A well-known benefit within the German tax law is the benefit that a taxpayer receives for his children, given his residence is on the

German national territory, gets for his children. The amount of money someone will be granted depends on the number of children entitled for this particular benefit. The calculation follows a rather simple arithmetical formula. However, the clarification of the conditions that have to be fulfilled, requires more additional parameters. At this stage, it is important to mention, that the modeling and description of the interpreted law only serves illustrative purposes and does not claim to fully reflect the content of the law or to be legally binding.

The first pre-requisite is, that a taxpayer[2] must have his permanent residence[3] on the national territory of Germany. German citizens can, by law, have only one permanent residence. Living abroad disqualifies him from retrieving child benefit[4]. A set of one or more children is required for which the taxpayer can claim child benefit. These children must not necessarily be his own children, but can also be the children of the spouse, grandchildren, or foster children. In addition, each child has an employment[5], which is either unemployed, job-seeking, education, in a so-called interim period, or in a voluntary social year. There are also other forms of employment in the German tax law, but they do not play a role in determining whether a child is qualified for child benefit or not. A child has a date of birth and also the information of whether it has a disability. This information is formalized in the semantic model of the child benefit regulation of the German tax law (§§62 - 66).

The semantic model of the situation as described above is shown in a box-and-lines diagram, following the UML notation for class diagrams, as shown in Figure 4.2. The model is subdivided into four different types:

- Taxpayer
- Residence
- Child
- Employment

Each of these types has individual attributes reflecting the information required to determine the resulting child benefit. For example, the taxpayer has the attribute “name”, which is of the type “String”. Figure 4.2 also shows the relations between the types. The relations do not only have names, indicating the semantics of the association, but also information about the multiplicities. The “claimChildbenefit” association connects the taxpayer and the child and is of the type one-to-many (1...n). The semantics can be interpreted as follows: each child has exactly one taxpayer to which it is associated and a taxpayer can claim child benefit for an arbitrary number of children.

Until now, only the static parts of the semantic model have been discussed. These parts are common in classical knowledge engineering and can also be represented using semantic web technologies, such as OWL. The remaining piece is of course the representation of the executable semantics. This is expressed in so-called derived attributes. The taxpayer type also has two attributes, which are separated from the “name” attribute with an additional line, namely

[2]dt. Steuerpflichtiger
[3]dt. Wohnort
[4]dt. Kindergeld
[5]dt. Beschäftigungsverhältnis

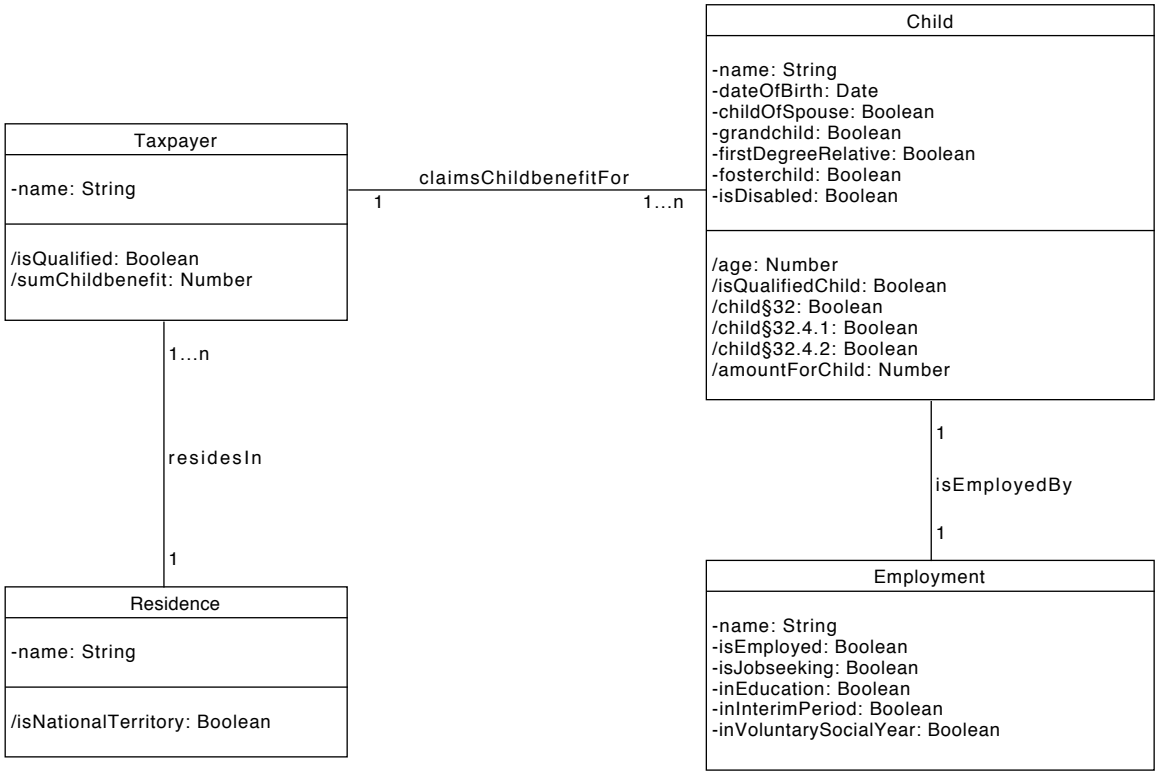


Figure 4.2.: Illustration of the German child benefit regulation in a semantic model.

"isQualified" and "sumChildbenefit". The former is of the type Boolean, the latter one of the type "Number". Contrary to the "name" attribute, these two attributes are indicated with a '/' (slash) instead of a '-' (minus). This notation is used for derived attributes. The two types child and residence also have derived attributes, as depicted in the diagram. A more detailed discussion about derived attributes and their formalization is provided in Section 4.2.6.

Based on this illustration, the next Sections 4.2.3 – 4.2.6 introduce the logical calculus to formalize the executable semantics.

4.2.3. Types

Based on the foundation of the German tax law on regulating the claim for child benefit, the logical calculus represents four different types:

$$Taxpayer \tag{4.1}$$

$$Child \tag{4.2}$$

$$Residence \tag{4.3}$$

$$Employment \tag{4.4}$$

Types represent the TBoxes[6] in the model-based reasoning approach. They structure the domain by describing the concept hierarchies. The instantiations of these types are called ABoxes[7], and are expressed as follows:

$$\begin{aligned} t : Taxpayer &\implies t \text{ is instance of } Taxpayer \\ c : Child &\implies c \text{ is instance of } Child \\ r : Residence &\implies r \text{ is instance of } Residence \\ e : Employment &\implies e \text{ is instance of } Employment \end{aligned} \tag{4.5}$$

Consequently, ABoxes are concrete instances of the TBoxes. They instantiate the abstract types and reflect facts or evidence within the model. For example, Equation 4.5 defines an ABox 't' which is of the type 'Taxpayer'. ABoxes and TBoxes reflect the structural properties of the model, such as attributes and relations.

4.2.4. Attributes

Types can have attributes, thus allowing the specification of properties and states. Figure 4.2 already introduced different types and their attributes. Moreover, attributes have different data types. The implemented approach supports seven different data types, namely:

- Integer
- String
- Boolean
- Date
- Float
- Double
- Link

While the first six data types in this list follow the classical semantics as in programming languages, the 'Link' data type reflects the relation between two types. A formalization of the attributes that exist within a type looks as follows:

[6] terminological box
[7] assertional box

$$\begin{aligned} Taxpayer.name &: String \\ Child.dateOfBirth &: Date \end{aligned} \tag{4.6}$$

The '.' notation indicates the access of an attribute that belongs to a given type. The attributes that are specified within the TBox exist in the corresponding ABox. Consequently,

$$\begin{aligned} Taxpayer.name : String &\implies t.name : String \\ Child.dateOfBirth : Date &\implies c.dateOfBirth : Date \end{aligned} \tag{4.7}$$

Given this specification, the following assignments are allowed:

$$\begin{aligned} t.name &= \text{"Daniel"} \\ c.dateOfBirth &= \text{"05/12/2010"} \end{aligned} \tag{4.8}$$

Whereas an assignment of the variable $c.dateOfBirth$ as follows would not be allowed:

$$c.dateOfBirth \stackrel{!}{=} \text{"Daniel"} \tag{4.9}$$

This would lead to an "incompatible types" error, since the types of date and string do not match. In addition to types and attributes, relations are required, which will be introduced in the next section.

4.2.5. Relations

The types are related to each other, which needs to be formalized. Therefore, the model contains three different relations (see Figure 4.2). The relation between a taxpayer and his child (4.10), between a taxpayer and his residence (4.11), and finally between a child and its employment (4.12).

$$\begin{aligned} &claimsChildBenefitFor \subseteq Taxpayer \times Child : \\ &(t, c) \in claimsChildBenefitFor \implies Taxpayer\ t\ claims\ child\ benefit\ for\ Child\ c \end{aligned} \tag{4.10}$$

$$\begin{aligned} &residesIn \subseteq taxpayer \times residence : \\ &(t, r) \in residesIn \implies Taxpayer\ t\ resides\ in\ Residence\ r \end{aligned} \tag{4.11}$$

$$\begin{aligned} &isEmployed \subseteq child \times employment : \\ &(c, e) \in isEmployedby \implies Child\ c\ is\ employed\ by\ Employment\ e \end{aligned} \tag{4.12}$$

The relations have been formalized using binary connectives, or predicates, of the form (a, b), whereas a and b are types. The calculus does not allow n-ary (for $n \geq 3$) predicates to specify relations.

Using types, attributes, and relations it is possible to semantically model ontologies for self-contained domains. The modeling approach supports a very flexible interpretation schema since it does not make pre-assumptions or forces users to use templates or boilerplates. Based on the UML notation, it should foster the creation and modeling of decision structures in a coherent and interpretable way.

The remaining components are so-called derived attributes, which are introduced in the next section.

4.2.6. Derived Attributes

Derived attributes are integrated on top of the existing modeling elements: types, attributes, and relations. As discussed in Section 4.2.4, types can have different attributes, such as name, birth date, etc. These atomic attributes can be facts or evidence given a formalized decision structure.

Derived attributes are inferred from those atomic attributes and are expressed with a formalized Domain-Specific Language (DSL) (see Section 4.2.7). Derived attributes are expressions, containing the required information on how the attribute is determined. A simple example is shown the following Equation 4.13:

$$c.age = TODAY - c.dateOfBirth \tag{4.13}$$

The Equation describes how the derived attribute "age", which plays a central role within the determination of child benefit, is defined. The "age" attribute is a classical example, since the provision of "age" is not necessary if the date of birth is already known to the system. Using a trivial mathematical operation, it is easy to determine a child's age.

The following Equations 4.14 – 4.17 formalize the remaining derived attributes for the type *Child*.

$$\begin{aligned} c.isQualifiedChild = c.child\S 32 \\ \wedge (c.childOfSpouse \\ \vee\ c.grandchild \\ \vee\ c.firstDegreeRelative \\ \vee\ c.fosterchild) \end{aligned} \tag{4.14}$$

The Equation above logically links the individual preconditions that would qualify a child for the benefit. Thereby, the logical connections are either between atomic attributes, or again refer to another derived attribute. For example, this is the case in Equation 4.14. While, the attributes $childOfSpouse, grandchild, firstdegreerelative, fosterchild$ are atomic attributes, $c.child\S32$ refers to another derived attribute, which is defined in Equation 4.15.

$$\begin{aligned} c.child\S32 = c.age < 18 \\ \vee\ c.isDisabled \\ \vee\ c.\S32.4.1 \\ \vee\ c.\S32.4.2 \end{aligned} \tag{4.15}$$

Equation 4.15 formalizes the claim regarding article 32 of the German tax law. It connects the arithmetical conditions $c.age < 18$ with the propositional elements $c.isDisabled$ using straight forward first-order logic. And again, the equation refers to two more derived attributes, which are formalized in Equations 4.16 and 4.17.

$$\begin{aligned} c.\S32.4.1 = (c.age > 18 \wedge c.age < 21) \\ \wedge\ \neg e.isEmployed \\ \wedge\ \neg e.isJobseeking, \\ \text{for } (c,e) \in isEmployed \end{aligned} \tag{4.16}$$

$$\begin{aligned} c.\S32.4.2 = \ (c.age > 18 \wedge c.age < 25) \\ \wedge(e.inEducation \\ \vee\ e.inInterimPeriod \\ \vee\ e.inVoluntarySocialYear), \\ \text{for } (c,e) \in isEmployed \end{aligned} \tag{4.17}$$

The Equations (4.14) – (4.17) specify the different conditions that are defined by law qualifying a child to be considered in the calculation for child benefit. The claim can arise from different articles in the tax law. This can be represented with the formalization. The example stated above reflects the different sections, from which the decision structure was derived (interpreted), using a naming convention. The example also shows how nesting can be used to reflect the structure of the law, i.e., sections, in addition to its semantics.

Now the set of children has to be determined as the basis on which the calculation of the concrete value, i.e., the amount of money, of the child benefit for a taxpayer can be carried out.

$$\begin{aligned} C^t = \{c \in Child \mid (t,c) \in claimsChildBenefitFor \\ \wedge\ c.isQualifiedChild\}\ for\ t \in Taxpayer \end{aligned} \tag{4.18}$$

$$t.isQualified = \ r.isNationalTerritory,\ (t,r) \in residesIn \tag{4.19}$$

The Equation (4.18) defines a set C^t consisting of all children for which a taxpayer t claims child benefit. The additional constraint does exclude those children that are not qualified for child benefit. Equation (4.19) ensures that the taxpayer lives on the national territory.

$$amountForChild(j) = \begin{cases} 190 & if\ 1 \leq j \leq 2 \\ 196 & if\ 3 \leq j \leq 4 \\ 221 & if\ 5 \leq j \end{cases} \tag{4.20}$$

$$t.sumChildbenefit = \begin{cases} \sum_{j=1}^{j \leq |C^t|} amountForChild(j) & if\ t.isQualified \\ 0 & if\ \neg t.isQualified \end{cases} \tag{4.21}$$

The remaining two Equations (4.21) and (4.20) determine the amount of the child benefit, based on the number of eligible children.

This section showed how a semantic model can be enhanced with an additional layer of executable semantics. It relies on arithmetical and functional logic, as well as on the components of the model. The model-based approach allows end-users to flexibly organize the decision structure with an additional level of abstraction, that is introduced with the knowledge engineering component from the ontologies.

It was illustrated how the two approaches, namely the knowledge structuring and the logical and arithmetical reasoning interact and complement each other. Until now, most approaches focused on either the knowledge structuring aspect by formalizing legal domains in various level of details and abstraction (e.g., upper- vs. lower-ontologies), or on the logical part (e.g., temporal or defeasible logics). Whereas the former lack of expressive rule engines that allow the specification of dependencies and constraints beyond description logic, the latter one lacks qualitative expressiveness in terms of structuring the semantic model within which the logical derivation is performed.

4.2.7. MxL: Model-based Expression Language

The MxL as a type-safe Domain-Specific Language (DSL) was developed to support reasoning within a generic meta-model-based information system that can be accessed via a REST API (see Reschenhofer et al. 2014, 2016).

According to Reschenhofer et al. (2014), MxL version 2 was influenced by the Object Constraint Language (OCL) and Microsoft's Language Integrated Query (LINQ), and it integrates the following main properties:

Functional programming: Expressions are the essence of MxL. Consequently, it follows the declarative programming paradigm.

High-order functions: MxL supports the development of high-order functions, i.e., functions that allow other functions as parameters or return a function as a result.

Type-system: MxL is strongly typed. Validity of expression regarding the proper use of data types is performed ex-ante.

Dynamic binding: Look-up and evaluation of attributes and methods are performed at runtime.

The meta-model-based information system incorporates the MxL for defining executable semantics based on a semantic model. MxL empowers end-users to apply simple (e.g., arithmetic) and higher-order functions (e.g., query operations), and to compose them as complex and nested expressions. In addition, it allows the access to methods and operations implemented in Java and can easily be extended by additional operators and functions.

An important property of MxL is its type-safety: It ensures that expressions are valid regarding their static semantics and thus supports end-users in defining consistent expressions with respect to the user-defined semantic model. Furthermore, MxL's type-safety enables the system to resolve dependencies between expressions, which in turn enables an automated adoption of expressions if referenced elements of the semantic model change.

In order to enhance the usability of MxL, we implemented helpful UI features, e.g., syntax highlighting for better readability, auto-completion including elements of the semantic model, and error localization in case of syntactic and semantic errors.

In the following, two listings show concrete examples of MxL expressions:

```
1 (this.'#age' > 18.0 and this.'#age' < 21.0)
2  and
3 (not this.'isEmployedby'.isEmployed or not this.'isEmployedby'.isJobseeking)
```

The listing above is the implementation of Equation 4.16 within the context of *Child* (this). The usage of the "this" operator (Lines 1 and 3) indicates the access of attributes in the context of a child instantiation. Line 3 also shows the access of the associated employment object through the relation "isEmployedby". Within the employment object, the isEmployed and the isJobseeking attributes are accessed (both of the type Boolean) and evaluated within the logic expression.

```
1 if not this.'isQualified' then 0
2 else
3   let betrag = (children: Sequence) =>
4     if children.count() <= 2 then children.count() * 190
5     else if children.count() = 3 then 2 * 190 + 1 * 196
6     else 2 * 190 + 1 * 196 + (children.count() - 3) * 221
7   in betrag(find(Child).where(c => c.'isQualifiedChild'
8                           and c.claimsChildBenefit.any(c => c = this)))
```

The listing above is the implementation of Equations 4.20 and 4.21 within the context of *Taxpayer* (this). If the taxpayer is not qualified for child benefit, the function returns 0. Otherwise, the calculation is performed on the set of children. The arithmetic logic follows the description in the tax law: For the first two children, 190 EUR (per child) are paid, the third

child is rewarded with 196 EUR, and for every child after the third the taxpayer receives 221 EUR. This illustrates the usage of the MxL language and how it is integrated into the semantic and executable modeling for child benefit.

The next section will elaborate on the different components for the model-based reasoning framework and the basic requirements to support the process of import, analysis, interpretation, and applications. It also describes the different components that have been implemented for the decision support system and what the user interface looks like.

4.3. Design of a Model-based Reasoning Framework

This section describes the conceptual framework that is required for support during the import, analysis, interpretation, and application of model-based reasoning. It constitutes the foundation for implementations and is subdivided into the elicitation and specification of requirements (see Section 4.3.1), the components of the decision support system (see Section 4.3.2), the extension within the system architecture (see Section 4.3.3), the meta-model-based information system (see Section 4.3.3), the interactive modeling interface (see Section 4.3.3), and the knowledge acquisition component (see Section 4.3.3).

4.3.1. Requirements

The main requirements were elicited and described in Waltl et al. (2017c). Therein, the requirements are classified into the four different phases along the process: import, analysis, interpretation, and application.

Import

Requirement 1: **Flexible import structure**
The baseline for the analysis and interpretation is the consideration of literature and textual documents (laws, judgments, contracts, commentaries, etc.) that is available in different formats (XML, HTML, PDF, etc.).

Requirement 2: **Mapping and indexing legal data**
The legal literature has to be indexed and mapped to a data model that does not only preserve the content, i.e., text and meta-data, but also structural properties, such as references and nested content.

Analysis

Requirement 3: **Preserving textual representation**
Enables users to access the content, i.e., legal literature. The visualizations of legal literature have to show the structural information, such as nestedness and links between articles and documents.

Requirement 4: **Collaborative creation and maintenance of patterns**
The creation, refinement, and deletion of the required pattern definitions should be done collaboratively in the application, so that different users are able to share their knowledge and contributions.

Requirement 5: **Management of pattern descriptions and linguistic model**
Support of the full lifecycle of the pattern specifications, namely creation, refinement, evaluation, and maintenance.

Requirement 6: **Automated pattern detection**
Automated identification of basic, linguistic, and semantic entities, e.g., obligations, prohibitions, and permissions, through data and text mining components.

Requirement 7: **Reuse of existing NLP components**
Building of NLP pipelines that allow the easy reuse and sharing of highly specified software components for NLP.

Requirement 8: **Evaluation of annotation quality**
Possibility to view the annotations, to examine precision and recall manually, or to export this information to compare against a manually tagged corpus.

Requirement 9: **Manually annotating and commenting legal texts**
Users should be able to manually add relevant semantic information and comments to the legal literature.

Requirement 10: **Storing annotations**
The system should permanently persist and index the automatically determined and manually added annotations.

Interpretation

Requirement 11: **Creation of semantic and executable model elements**
Stepwise definition of model elements (types, attributes, relations, and derived attributes) for semantic and executable models.

Requirement 12: **Lifecycle support for semantic models**
Defining, maintaining, and persisting of semantic model elements, such as types, attributes, and relations.

Requirement 13: **Lifecycle support for executable models**
Defining, maintaining and storing of executable model elements, such as derived attributes.

Requirement 14: **Correspondence of model elements with text phrases**
Creation of connections between model entities and the relevant, i.e., interpreted, text. Thereby, various parts of the text should be linkable to model elements, including words, phrases, sentences, sections, and documents.

Requirement 15: **Expressing executable semantics with a DSL**
Specification of the operations and executable semantics of derive attributes with a functional expression language supporting logical and algorithmic reasoning.

Application

Requirement 16: **Access to existing models**
Viewing and exploring semantic and executable models to retrieve the results of prior interpretation processes.

Requirement 17: **Application of models**
Providing facts and executing the defined models through forms.

4.3.2. Components of the Model-based Decision Support System

The requirements are the base line of the decision support system in order to fully comply with the formalization according to the schema as described in Section 4.2. Legal informatics has dealt with the construction of software systems to support legal reasoning ever since. The idea of letting an algorithm decide on low-level and data-driven decisions is still highly attractive in the legal domain (see Ashley 2017).

Jandach (1993) analyzed different notions of LES with a particular focus the concepts and characteristics that address LES for civil law systems, more specifically the legal system in Germany. Several attempts have been made to implement decision structures, arising from German legal texts, into rule-based systems. However, no attempt has been made to formalize German laws using a model-based, i.e., ontological approach, with a reasoning engine that enables users to define expressions and to infer knowledge using propositional logic, first-order predicate logic, and arithmetical logic alike.

Jandach studied different parts of LES and described the main components that form a minimal set of a viable expert system. These components are: knowledge base, inference engine, explanation component, knowledge caption component, and dialog component. Based on this, we extended the idea and developed the conceptual model as shown in Figure 4.3.

The system's components can be classified into three different groups, namely a model store, a model execution component, and an interaction component. Each group is implemented by multiple different software components. The following sections briefly introduce these with their responsibilities in the model-based decision support system.

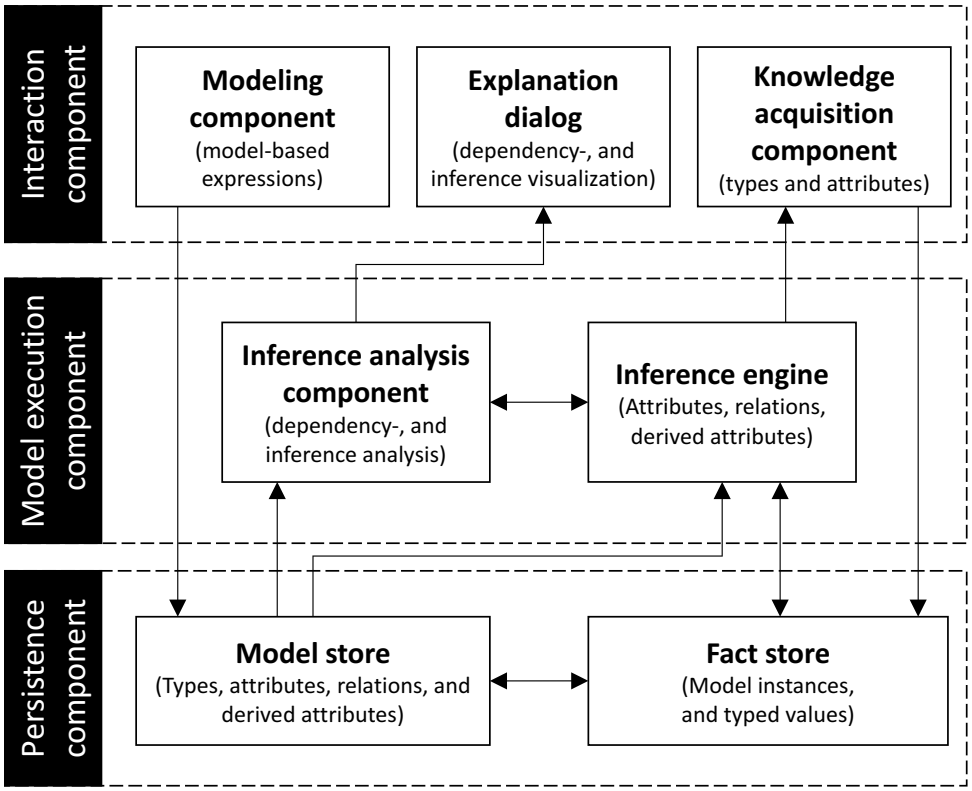


Figure 4.3.: Overview of a model-based reasoning system and its interactions grouped into three components: model store, model execution, and interaction component.

Interaction Component

The improvements of modern software systems with regard to user experience can be considered as one of the main successes of software engineering in the last decade. End-user development and human-centered design thinking have become established methodologies in designing and implementing software systems. Beside the provision of bare functionality, such as logical reasoning, LES need to offer user interfaces that do not overexert users. Instead it will be the challenge for the legal informatics domain to enable end-users to use LES and to leverage the full potential that they offer.

In our system, the modeling component is separated from the knowledge acquisition component and the explanation dialog:

Modeling component: Users, e.g., legal data scientists or legal knowledge engineers, are supported in the creation of the model with its attributes and relations by a modeling component that renders a graphical view of the model. It offers functionalities to create, update, or delete types of a model. The same operations are also offered for attributes and derived attributes.

In this process, the users are supported with a graphical user interface that runs entirely

as web application. The modeling component also allows to instantly access different legal documents and their textual representations.

Knowledge acquisition component: The provision of facts is carried out in a separate component, which has been exclusively designed for this purpose. Types, relations, and attributes, e.g., describing a legal case, can be added by the users. Due to the strongly typed expression language, attributes are already type-checked in the interface as they are input. It is not possible to insert incompatible types, such as text if a number is required. This should decrease the error rate and allow for consistent input. The knowledge acquisition component shows the inferred values of the derived attributes.

The component was developed within a web application, which allows the usage of modern front-end technology and visualizations to lower the barrier of providing facts.

Explanation dialog: To reconstruct the conclusion that was inferred by the system, users need a function to provide them with information about the reasoning procedures. This explanation dialog visualizes the information from the inference analysis component. Given a particular derived attribute, the user receives information about the underlying MxL expression and the abstract syntax tree showing the different data values and operators that contribute to the overall result.

The analysis of decisions and inferences is of high value within the domain of decision support systems for the legal domain. This increase of transparency of the reasoning process increases the trust and reliability of the system and allows for detailed error analysis.

Model Execution Component

The model execution components are built on top of the model storage and accesses the database of facts, i.e., instantiation of types with concrete values for the available attributes, and the database containing the information about the schema, i.e., types, attributes, and relations.

Inference engine: The reasoning on the given facts considering the formalized rules requires access to the database of facts and the storage holding the information about the expressions required to determine the derived attributes. The inference engine is developed in a separated service, which can be consumed via a REST API. The inference engine is developed in Java and retrieves data from the meta-model-based information system. The MxL was designed using "Beaver - a LALR Parser Generator"[8]. It is strongly connected to the persistence component, which is a meta-model-based information system (see Section 4.3.3).

The inference engine offers end-users to define semantics of derived attributes in functional expressions, and allows the expression of first and second order logic as well as the definition of complex queries (projection, selection, and transformation).

Inference analysis: Closely connected to the inference engine is the inference analysis component. This component allows the inspection of complex expressions. It allows for the retrieval of the Abstract Syntax Tree (AST) information of an expression and it also enables users to

[8] http://beaver.sourceforge.net, accessed on September 3, 2018

obtain an overview on the provided and derived facts in an object diagram-like visualization (see Section 4.4).

The component offers functionality to view complex data flows, based on the input parameters, to inspect the resulting derived attribute. This enables the inspection of direct and indirect influences that types, relations, and especially attributes have.

Persistence Component

The model storage component contains the definition of the model, i.e., ontological description, and the facts provided by the end-user.

Model: A model is described by its types with their attributes, i.e., schemes, and the relations between the types. Our implementation differentiates between two types of attributes: atomic attributes and so-called derived attributes. Atomic attributes consist of concrete values and have a basic data type, e.g., number, date, text, enumeration, Boolean, and sequence. In contrary, derived attributes are expressed as rules, formalized in MxL, which is a strongly typed and functional DSL.

Facts: The instantiation of a model is done through the provision of facts. Those facts are stored as explicit records in the model store. Each model instance has a unique identifier and name which is used for unambiguous identification. An instance does not need to assign a value to each attribute. The attributes are optional and null-value (empty attributes) are allowed.

The prototypical implementation is based on a meta-model-based information system to provide the model storage component. This information system persists all the information about the scheme of the ontology, i.e., model, as well as the instances, i.e., facts, of a concrete model. The system allows the formalization of different models that are logically separated into disjoint workspaces.

4.3.3. Extension of the System Architecture

The implementation of the model-based reasoning system has an impact on the overall architecture of the systems as introduced in Section 3.5. Figure 4.4 shows the extended system architecture. Details regarding the text analysis engine and the database and search-engine are omitted, as those are shown and discussed in Section 3.5.

The extension of the existing system impacted three different components. While one component was just extended, i.e., user interface, two new components have been added to the system, i.e., the modeling component and the execution and reasoning engine. The principle of having one harmonizing data access layer has been preserved. It encapsulates the access and data exchange between the user interface, the modeling component, and the reasoning engine, which is accessed as an external service.

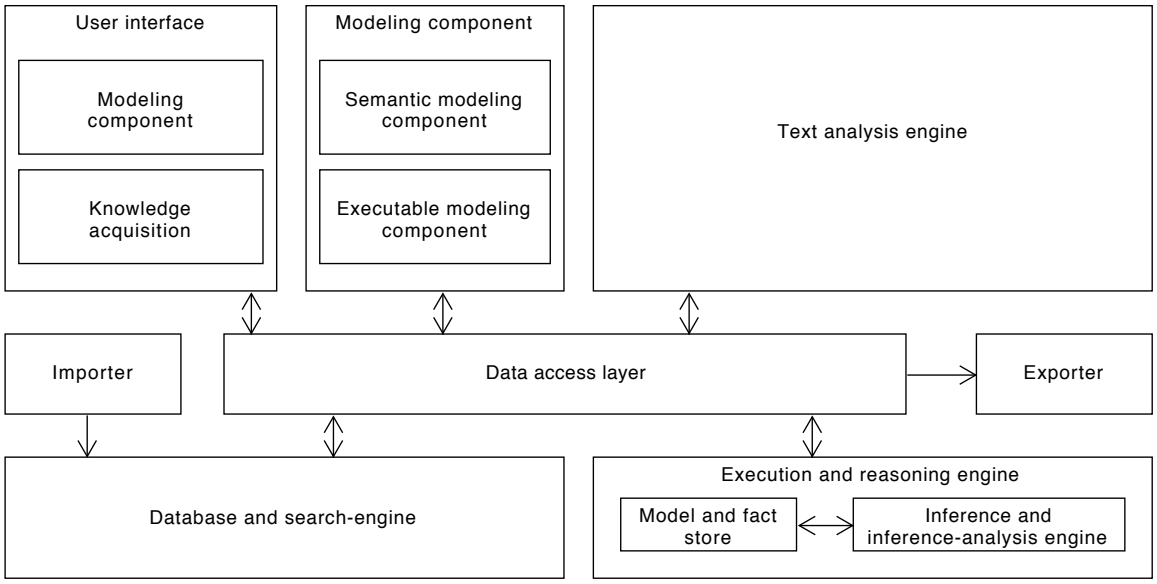


Figure 4.4.: Overall system architecture including components for model-based reasoning.

Knowledge Acquisition Component

The view is structured into three different regions as shown in Figure 4.5.

On the left side, an overview of the model is displayed. In this interactive diagram, the attributes are omitted to reduce the visual complexity. It shows the types and the relations among them. This model allows to select one of the types, which changes the form-based knowledge acquisition in the middle of the view. The canvas supports zooming and dragging.

In addition, below the model view on the left side, the actual instantiated model is visualized, in following an UML object diagram notation. A larger view of this particular screen view is shown in Section 4.4.1.

The middle part changes depending on the selected type and renders the available attributes, references, and derived attributes. Figure 4.5 shows a taxpayer "Bernhard" who only has one attribute, i.e., a unique name. He has two references: one to the residence type and one to the child type. In the example shown, there is one reference to the residence "München" and two references to the children "Isabel" and "Daniel". According to his derived attributes, the taxpayer is qualified for retrieving child benefit. In addition, the sum of child benefit, that he retrieves is 380. This form is automatically generated, depending on the selected type. Changing this selection changes the lists of available attributes.

The right part reflects the correspondence with the model type. During the creation of the model, the user can select different parts of an indexed documents, e.g., whole documents, single sections, or even single annotations, and attach them to a model type or attribute.

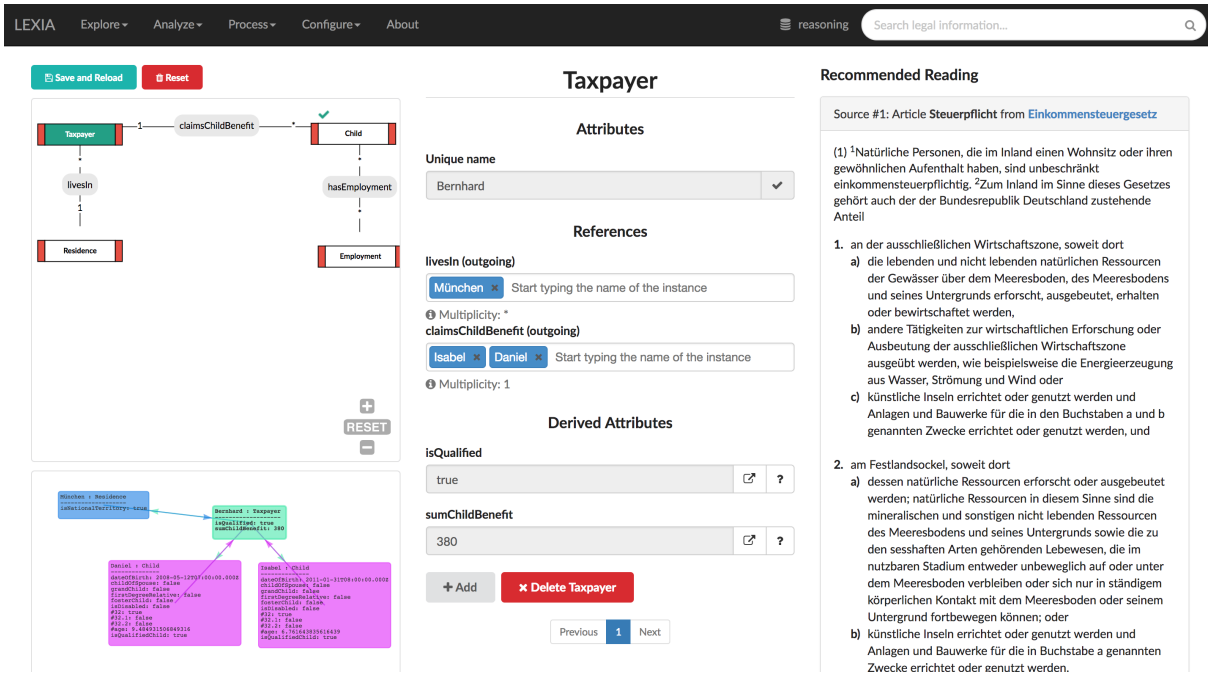


Figure 4.5.: Knowledge acquisition interface organized in three areas: model and instance view on the left, knowledge acquisition in the middle, and linked documents on the right.

Modeling component

The modeling component is separated from the knowledge acquisition component. It allows users to add new types and to assign attributes to them. The concept does not allow for attributes that are not assigned to any type, such as global variables. The user interface of the modeling component is shown in Figure 4.6.

The view is structured into a left and right part. The left part allows users to read documents that are already indexed in the system. The right part allows users to view and manipulate the model. Having these two views side by side should help users to analyze the text and immediately capture the interpretation result in the model. This activity should not require an interruption of the workflow by constantly switching between the document and the model.

An new type can be added using the control elements above the model canvas. More details about the types is shown in a pop-up, which appears by clicking on the type object in the model canvas. An excerpt of a pop-up is shown in Figure 4.7. It depicts the attribute view for the child type. In concrete, the attributes 'age' and '32.1' are visualized. Both are derived attributes, which is indicated by the 'MXL' selection of the control. For derived attributes, an editor is shown that allows the specification of the expressions. The editor supports auto-completion and syntax highlighting. The two equations shown in Figure 4.7 are 4.13 and 4.16. Each model element can be linked to the document (or to parts of it). These links are created during the modeling process. For attributes, the links can be added by clicking on the 'show link'

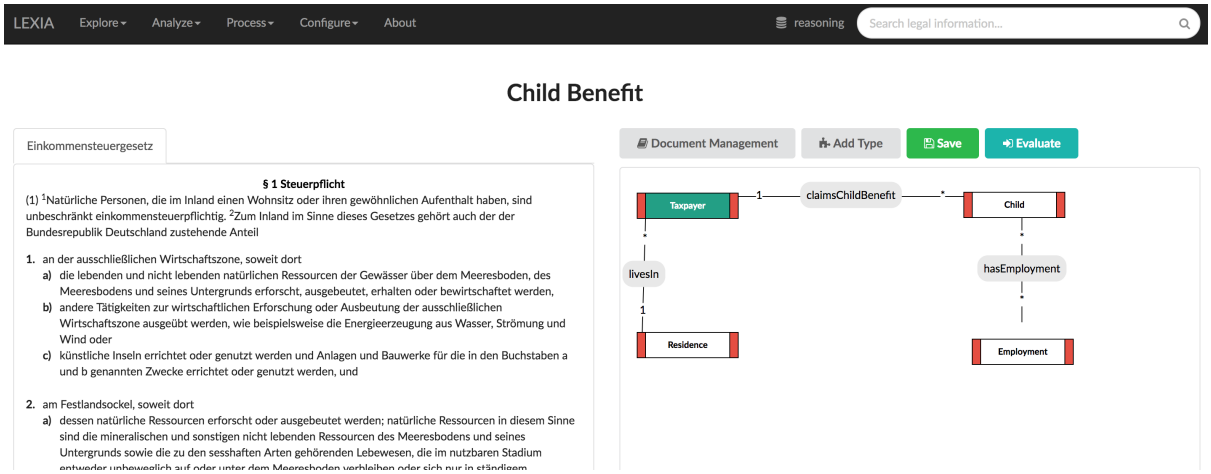


Figure 4.6.: Modeling interface organized into a document (left) and a model view (right).

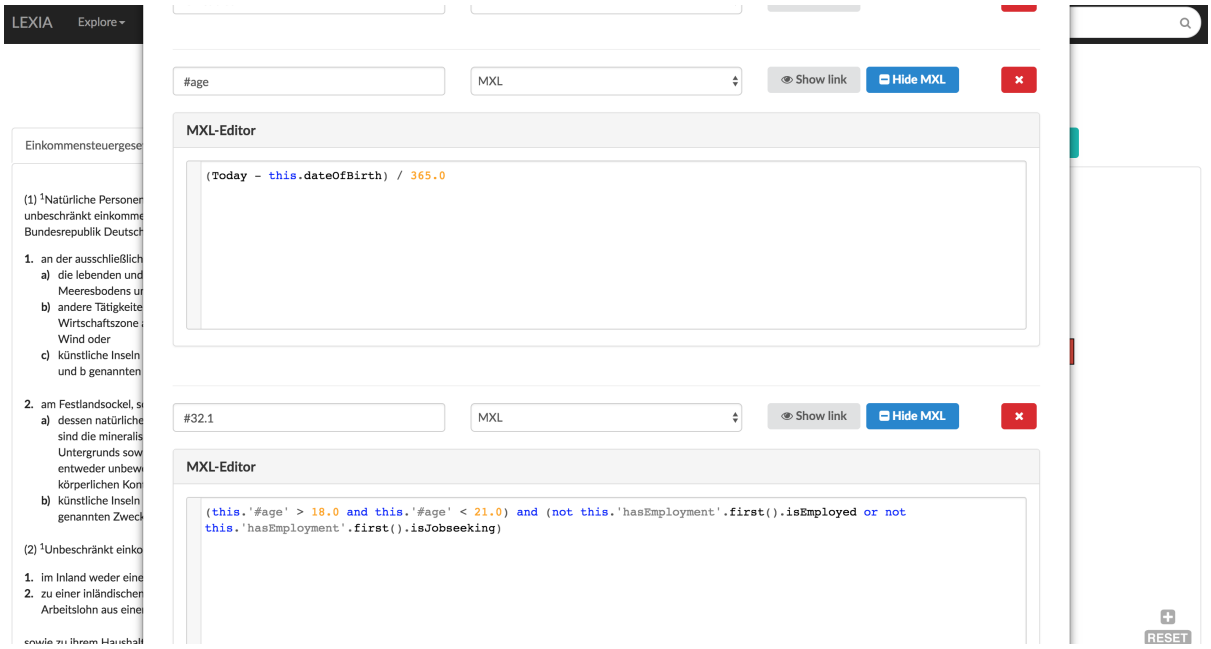


Figure 4.7.: Attribute manipulation within the modeling component.

button (see Figure 4.7). The linked documents are shown later on in the knowledge acquisition component to help end-users understand the modeled reasoning structure in detail.

The model is stored in the model store by initiating the saving process. During this process, the consistency of the model is automatically checked. Mistyped expressions or type failures are determined and the model is not going to be stored. This decreases the error rate and the usage of wrong models, but increases the effort for consistent modeling. The system does not check the plausibility and the semantical correctness of the model-based decision structure, but only its syntactical and well-formed model representation.

Execution and reasoning engine

The evaluation of the executable model elements, i.e., derived attributes, is done in a separate component, which was described by Reschenhofer (2017). The component is accessed via a REST API and is responsible for storing and persisting the model and instances of the model, and for evaluating MxL expressions. The model is serialized into a JSON format, which is created within the data access layer. The results of an evaluated expression are also wrapped within a JSON document and sent back to the data access layer of the implemented system.

This section will briefly explain how expressions are evaluated and which particular steps are performed to get a concrete value out of the expression. The measures are shown in Figure 4.8. A detailed explanation about the different steps and their implementation has been published by Reschenhofer (2013) and Reschenhofer (2017).

The expression is inserted as a plain string and is checked by the MxL scanner regarding its syntactical correctness. The scanner was created using a LALR parser generator[9]. The scanner is automatically created based on the specification of the grammar within the Extended Backus-Naur form. Based on the tokens, the MxL Parser transforms the expression into a data structure, which is called the AST. The AST is the formal representation of the expression and allows additional contextual analysis, such as type checking, which is performed in the subsequent step.

The MxL type checker accesses the model and fact store via a connector component. This component encapsulates the access of the storage, which is a document-based NoSQL database. In a first step the model information is retrieved. The model information is the specification of types, attributes with their type information, and the relations with their multiplicities. Based on this information, the type checker validates the different operations that should be executed on a meta-level and ensures well-formed expressions with regard to the type information.

Finally, the typed expression tree is forwarded to the evaluation engine. The evaluation component retrieves the information about the instances via the MxL connector. These facts have been provided by the knowledge acquisition component and are persistently stored. The facts are used as input values for the evaluation of the expression tree. The final result of the expression is determined by evaluating the leaves and subsequently aggregating the results of the subtrees.

[9]Beaver - a LALR Parser Generator, http://beaver.sourceforge.net/, accessed on September 3, 2018

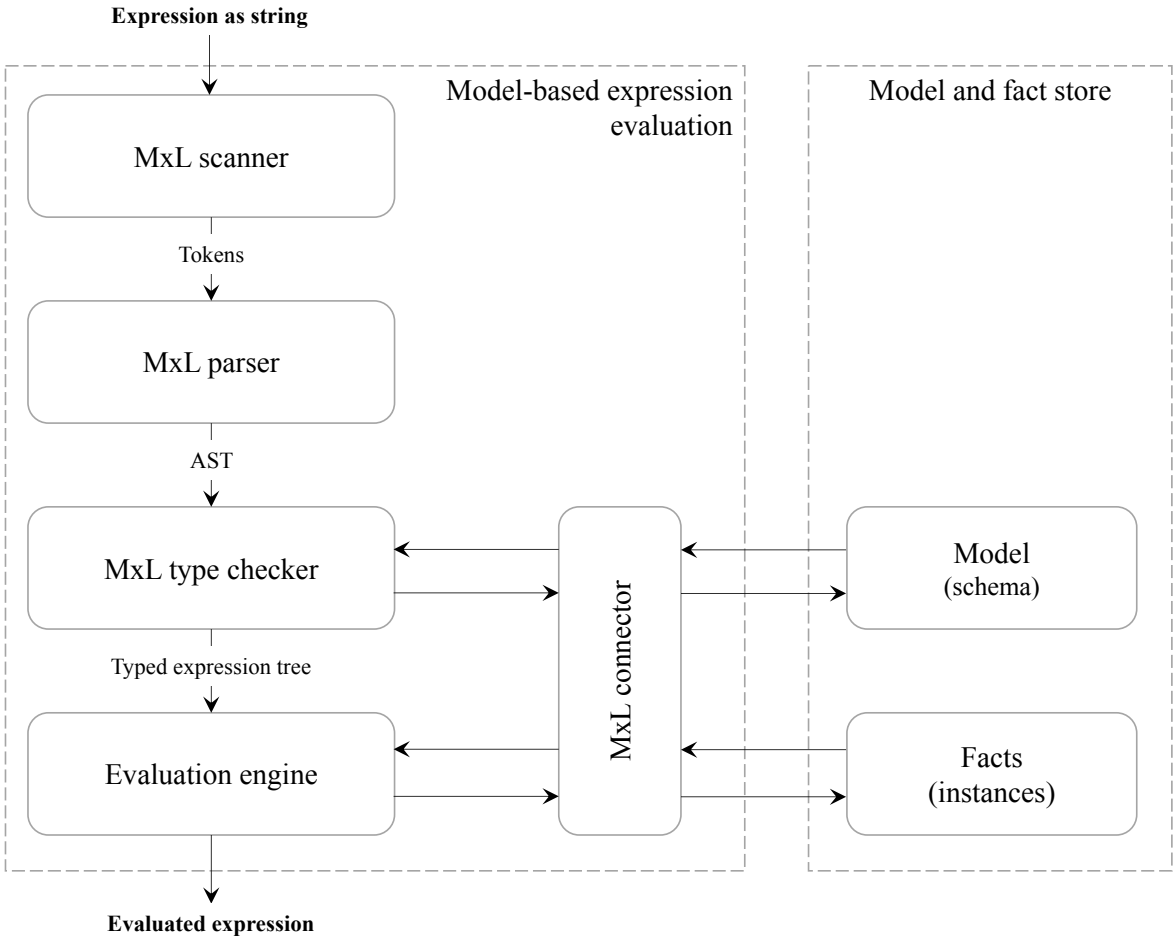


Figure 4.8.: The evaluation path of an MxL expression according to Reschenhofer (2013, p. 39).

4.4. Analysis and Explanation of Decisions and Decision Structures

Formalizing the decision structure allows for a thorough inspection and to automatically determine dependencies in attributes and types. The field of explainable artificial intelligence is becoming increasingly popular. Within legal decision making however, it has been important ever since. The components for the model-based decision support systems, namely the inference analysis component and the explanation dialog, have already been discussed in Section 4.3.2.

This section provides a more illustrative account of explanation components and how these provide additional information to understand and reconstruct automatically-made decisions. Thereby, four different components, namely an instances and fact view, an AST for dependency analysis, an explanation dialog component, and a data information flow inspection component, are introduced.

4.4.1. Instances and Fact View

Based on the model structure, it is possible to instantly combine the inserted facts and automatically visualize a given case. The inspection of the concrete instances with their attributes and relations with each other is another source of information and can provide additional insights into a case.

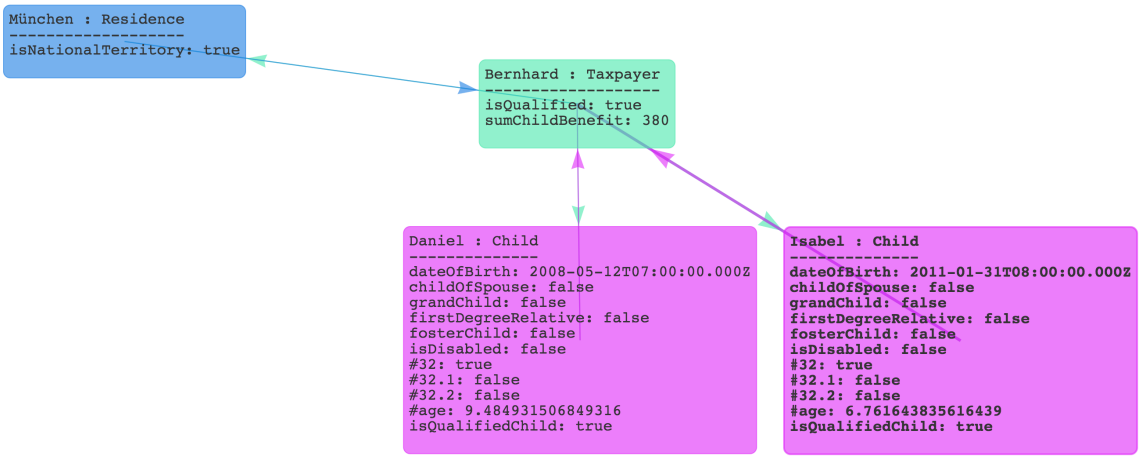


Figure 4.9.: Automatically and instantly created object diagram visualizing types, attributes, relations, and evaluated derived attributes from the model and fact storage.

Figure 4.9 shows an overview of a concrete case that was instantiated on the model for child benefit determination of the German tax law. Four different instances exist in it: one of the type "Residence" (blue box), one of the type "Taxpayer" (green box), two of the type "Child" (pink boxes). The notation follows the UML notation for object diagrams. The first line of each box indicates the instance name and the type. A line separates this general information about an instance from its attributes. The attributes are rendered as a list, whereas each attribute is assigned to a separate line. The lines contain the information about the attribute and separate the concrete value of the attribute within this instances with a colon ':'. For example, the taxpayer Bernhard has two attributes, of which the latter one indicates, that he is eligible to retrieve 380 (sumChildBenefit attribute). This can be reconstructed by analyzing the two assigned children Daniel and Isabel, who are both qualified for child benefit.

The relations between the instances are also visualized as lines with an arrow. The names of the relations are omitted.

4.4.2. Abstract Syntax Trees for Dependency Analysis

In order to examine the input parameters leading to a certain decision, i.e., value, the logical expressions can be analyzed automatically. Basically, this analysis is performed each time an expression is evaluated. However, the information about this analysis is not stored or persisted

for every evaluation. This would be unnecessary overhead. This information can be retrieved on request.

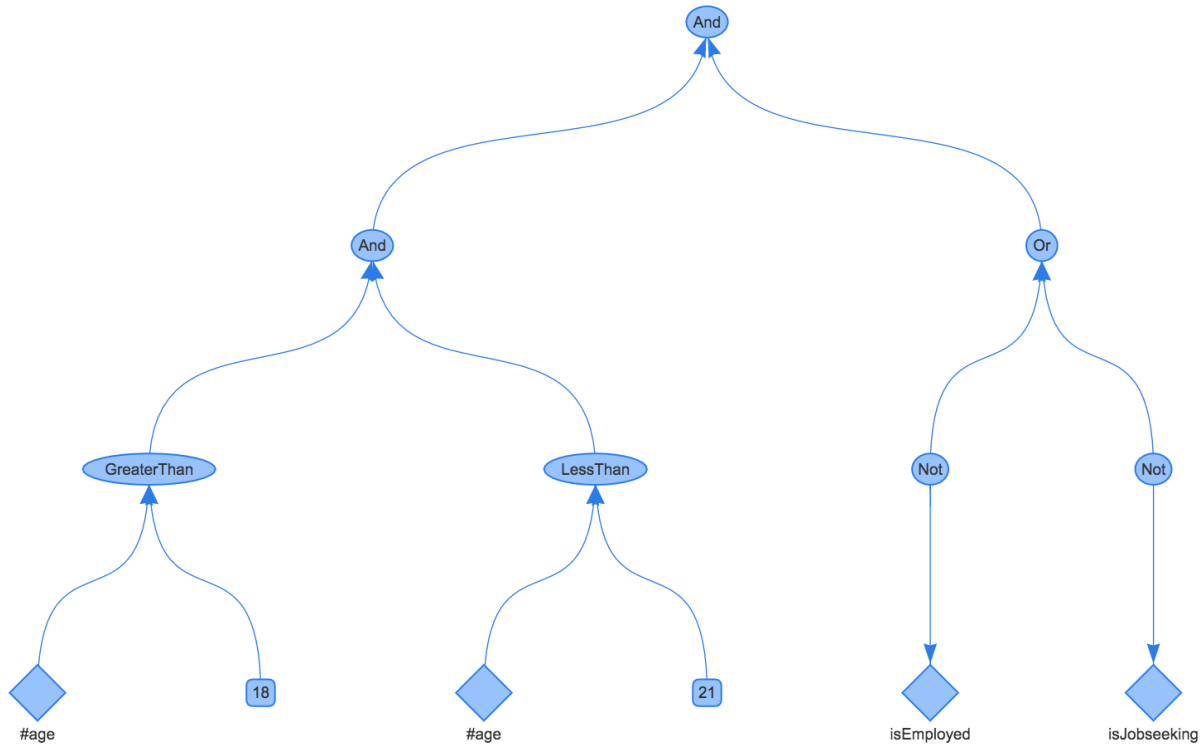


Figure 4.10.: Automatically determined AST for derived attribute §32.4.1 of the type child.

Figure 4.10 shows the AST for a derived attribute from the child type. The formal specification is provided in Equation 4.16. The tree consists of two main branches: a condition for the age of the child, which has to be between 18 and 21, and a logical operation on the employment and "jobseeking" attribute.

The illustration shows the resolution of the nodes and leaves, which reflect either conditionals and operations or attributes and constants. The attributes and constants are in the leaves, whereas the operations are the nodes. Consequently, the leaves show the attributes that influence the decision and the nodes reflect the logic and how values are processed.

The AST can be extracted during the evaluation process of an MxL expression. As shown in Figure 4.8 the AST is the intermediate result of the second evaluation step after the parsing. The rendering as shown in Figure 4.10 is carried out using a JavaScript[10] component. Section 4.4.3 will introduce the integration of this view into the user interface of the knowledge acquisition component.

[10]http://visjs.org/, accessed on September 3, 2018

4.4.3. Explanation Dialog Component

The explanation dialog is the integration of the information that is available during the reasoning process at the front-end. The information is requested from the inference analysis component and rendered so that it can be accessed and interpreted by end-users.

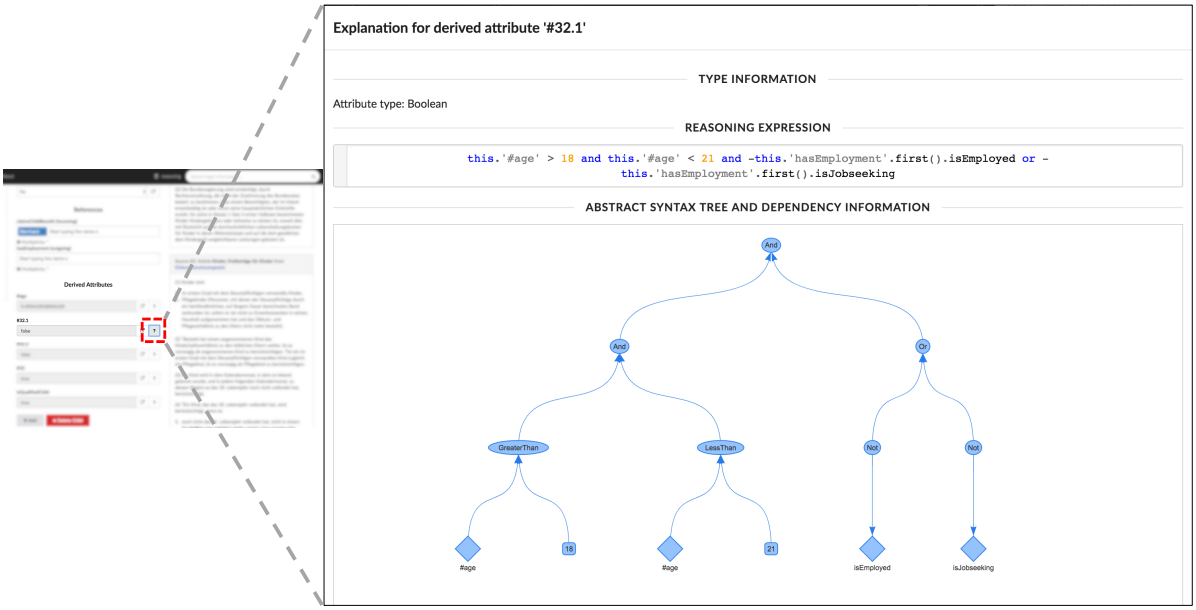


Figure 4.11.: Information on the decision structure: type information, MxL expression, and AST.

Figure 4.11 illustrates this integration within the knowledge acquisition component. Thereby, each derived attribute is enriched with an additional button "?" that opens the explanation pop-up for the selected derived attribute. The pop-up is initialized with the information from the inference analysis component. It consists of three different information fields: type information, reasoning expression, and abstract syntax tree and dependency information.

The type information is the data type from the evaluated value of an expression. It is determined based on the MxL expression. As the expression language is strongly typed, this information can provide a first insight on what to expect as the resulting value. The reasoning expression is the logical and arithmetical formula and expresses the formalization. This is the string as created during the modeling process. Finally, the AST allows the graphical inspection of the dependencies within a given expression.

This view provides information about its generation and allows backward inspection of how it was determined. It does not provide any insight of the attributes and decisions that are influenced by its value. This analysis is done by the data information flow, which is introduced in the next section.

4.4.4. Data Information Flow Inspection

The data information flow inspection is an additional view, that is determined given a concrete semantic model with executable elements. The inspection tracks the information flow between types and attributes. In contrast to the AST, it does not analyze the decision structure on the level of attribute, but qualitatively derives dependencies and how the information propagates through the network.

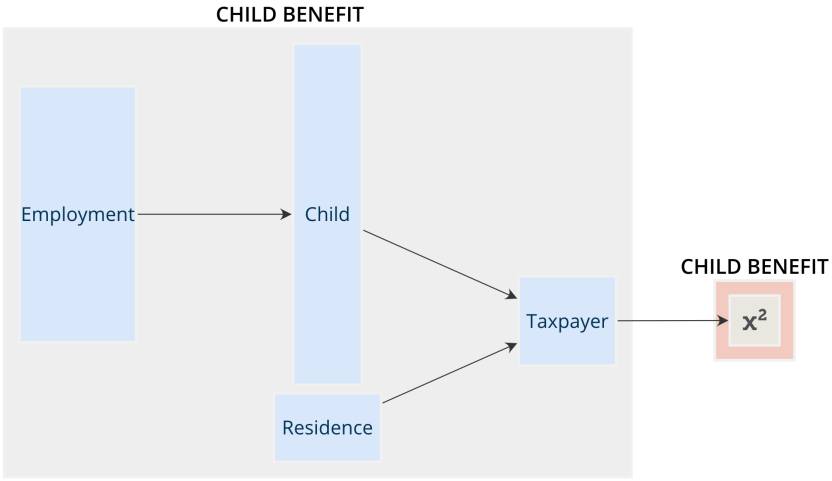


Figure 4.12.: Overview of the types and the high-level data flows between model elements.

Figure 4.12 shows an overview of the data flows within the formalized semantic model of the child benefit from the German tax law. It is automatically determined and contains four different model elements, indicated by blue rectangles: “Employment”, “Child”, “Residence”, and “Taxpayer”. The overview aggregates the attributes of a type and groups them together. The arrows between the types indicate how the information from one component is used within another component, i.e., input of a derived attribute.

The data flow inspection can also be performed on a detailed level, differentiating between all model elements that contribute to a final result: types, attributes, and derived attributes.

Figure 4.13 renders the automatically determined data flows between the model elements on a fine granular level. The blue rectangles still group different attributes and derived attributes, however, they are no long treated as black boxes, but as white boxes showing their internal structure.

The figure shows atomic attributes as blue rectangles with a gray border. For example, the type “Child” contains several atomic attributes, e.g., dateOfBirth, hasEmployment, etc. In addition, the derived attributes are rendered as light green boxes. Again, the arrows indicate the data flow direction and the reuse of atomic attributes within derived attributes. Trivially, blue boxes cannot have ingoing edges, as they are atomic. On the contrary, the green boxes, i.e., derived attributes, can have ingoing arrows from atomic and derived attributes. The already studied example of the attribute “§32.1” is in the center of the child object and accesses information from the derived attribute “age” and the “hasEmployment” attribute of the same instance, indicated

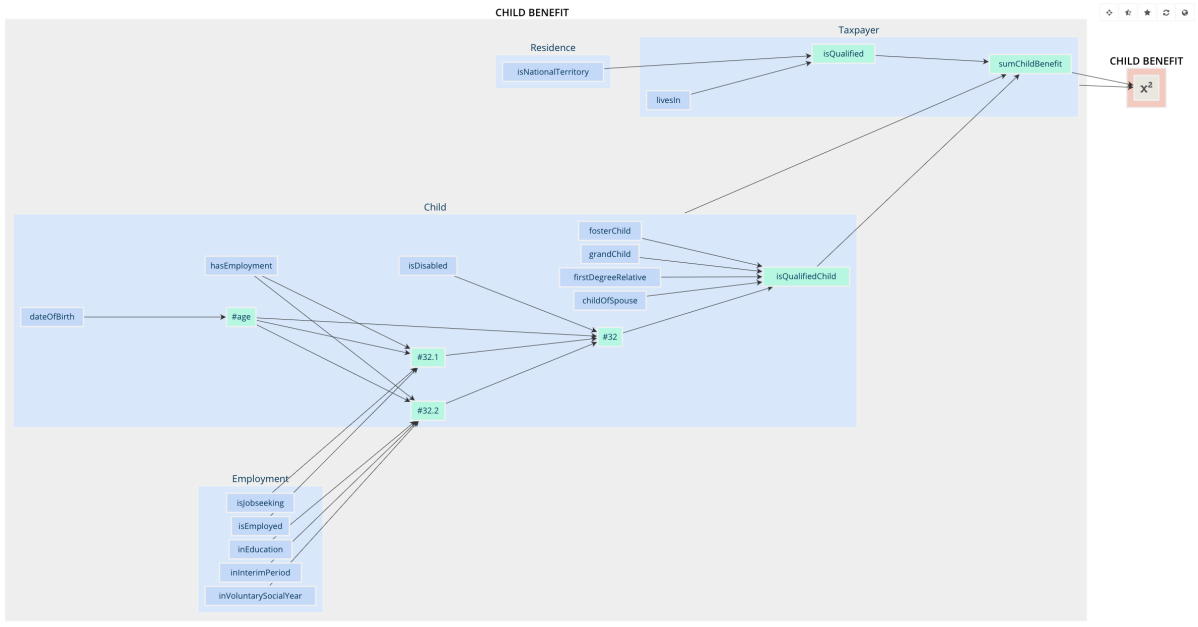


Figure 4.13.: Data flows with fine granular resolution on the level of types, atomic attributes, and derived attributes.

with the "this" keyword (see Figure 4.11). The attribute also accesses two attributes of the type "Employment", namely "isJobseeking" and "isEmployed".

Although this is not the case within the shown figure, derived attributes can also exist without having ingoing arrows. This would be the case if the derived attributes do not need any factual information to provide their results. A scenario could be the determination of days passed since a give date. In tax law, those deadlines are very common. However, the provided example does not contain such a derived attribute. Just like Figure 4.12, only the dependencies and potential influences are shown. This a qualitative inspection, as the concrete expressions and the logical formula are not inspected. In addition, this is a static analysis on the decision structure and does not allow for the analysis for concrete instances with real values, i.e., facts.

4.5. Summary

This chapter introduced a logical calculus to enable model-based reasoning on interpreted legal norms. Based on legal theory and the interpretation of statutory texts, a reference process was developed, which was divided into three different categories: activities, roles and services & tool-support. It discusses if and how automated semantic analysis of legal documents can support the interpretation of statutes and how these interpretations can be formalized into model-based decision structures. Once formalized, these structures can be applied to reason on decision structures, solely by providing the facts of a case. The decision structures do no longer need to extracted from the text.

Based on these foundations, 17 main requirements have been derived that need to be met in order to implement the reference process within a software system. They have been divided into four different groups reflecting the activities along the reference process:

1. Import
2. Analysis
3. Interpretation
4. Application

The basic idea of reasoning on models of the real-world, so-called ontologies, has already been investigated in the field of Legal Expert System (LES)s with success. The approaches either focused on either the knowledge engineering part or on the reasoning part, using standardized description logics. Description logics, however, are not intended to support reasoning over arithmetical expressions and are not designed to express the semantics of attributes of types. They are rather used in defining constraints and axioms of an ontology. This limitation was solved in the approach by introducing a Model-based Expression Language (MxL). The approach overcomes this limitation by harmonically combining

1. modeling of interpreted ontological knowledge, and
2. reasoning on ontologies with higher-order logical and arithmetical operations.

The approach was illustrated in a case study from the German tax law, namely by modeling the claim for child benefit. The model elements are either types, relations, attributes, or derived attributes. The latter ones are MxL expressions and are that are automatically determined using an inference engine. The inference engine is part of an overall system consisting of interaction components, model execution components, and a persistence component. Each of them has been implemented into the overall system.

The system does not only allow to formalize the models and to reason automatically, but it also enables end-users to analyze and inspect the decisions and decision structures. The inference engine is enhanced with an inference analysis component, that provides information for diverse explanation dialogs. These dialogs offer information on the instances and facts that are stored, about dependencies of derived attributes based on their Abstract Syntax Tree, and about the overall data flow within the model.

CHAPTER 5

Implementation

This chapter describes the main aspects of the implementation for an environment that allows to semantically analyze legal documents and to formalize the interpretation of statutory texts in executable model-based decision structures. The environment has been implemented as a web application and follows a classical model-view-controller pattern (see Section 5.1).

Within this framework, multiple different components are implemented or integrated, enabling a broad spectrum of analysis tasks. Following the technological principles of modularity and re-usage of software components, the system adapts the Apache UIMA with its Pipes & Filters architecture. The components are logically grouped in the text mining engine. Concrete implications for this architecture on the implementations are discussed in Section 5.2. Therein, a particular focus is set on the extraction of information and annotations from legal documents.

Extending the components of Apache UIMA by the integration of AML is discussed in Section 5.3. A particular focus is set on the data exchange between the text mining engine and the AML component. The active learning engine is described in detail, as well as the implemented query strategies, which are central components for differentiating the approach from supervised machine learning without query strategies.

Based on the considerations for the analysis of legal documents, the main aspects regarding the implementation of the model-based reasoning framework are described in Section 5.4. It follows the structure of decision support systems as introduced in the previous section, namely the persistence components, the model execution component, and the interaction component.

5.1. Collaborative Data Science Environment

To enable collaboration within the semantic analysis of legal documents and the modeling of executable model-based reasoning structures, the environment was implemented as a web application. The established Play Framework (2017) served as the basic web application stack.

5.1.1. Framework

The Play Framework is open source and follows the model-view-controller architectural pattern.

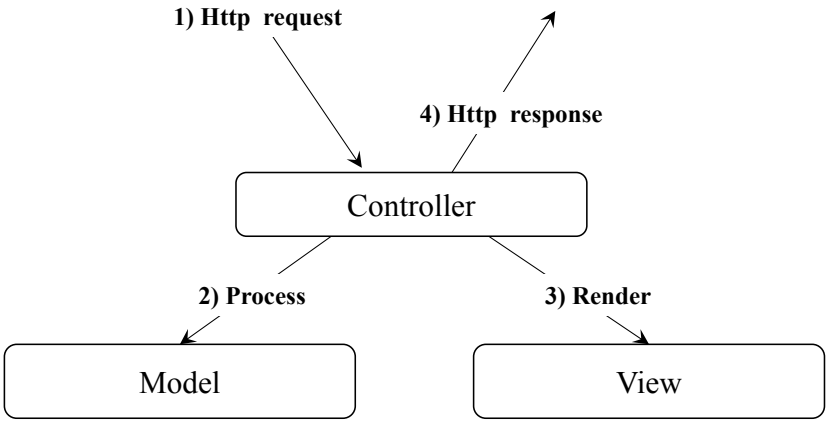


Figure 5.1.: Handling requests with the model-view-controller principle for the Play Framework.

This pattern splits the web application into two different tiers, namely a presentation and a model tier, whereas the presentation tier is additionally divided into two layers, namely controller and view layer.

Model

The model represents the domain-specific information on which the whole application operates. It is the base line for the performance of additional logic and implements the specific data model. The schema of the data model is implemented in the model layer, and the connection and mapping to the data storage are also maintained within this layer.

In addition to the connection of data storages, the model contains the business logic. In LEXIA, this means the whole processing pipeline and all the different components for the semantic analysis are part of the model layer. Therefore, the model layer is not only the data access layer, but also contains the functionality for the operations that are performed on the data.

View

The view component usually provides the interfaces to access the functionality provided by the model layer. In most cases, these are user interfaces to view the results of prior processing steps, or to initiate the processing of a particular data element, e.g., a law. It is common to provide more than one view for a model component.

In the LEXIA system, this is used for a separate view to visualize the annotated content of a document, or to initiate a processing pipeline for a selected document. In modern web applications, views are mainly created to be represented and rendered in web browsers that support various formats, e.g., HTML, XML or JSON.

Controller

The controller is a central layer connecting the user interactions or requests from outside the system with the model layer to create responses. The events triggering the controller are typically HTTP requests and are initiated by either user interaction or by algorithmic agents. Controllers extract the information from the requests, such as query string parameters and request headers, processes them, and forward the action to the model layer where the changes in the underlying data are executed.

The LEXIA system consists of several different controllers, e.g., ImportController, DocumentController, and ModelController, which allows the logical separation between different interactions and events that should be performed on the data. Consequently, the system can easily be extended to provide additional features. These must not necessarily be consumed by humans, but can also serve as service interfaces for other applications and frameworks as, for example, in service-oriented architectures.

5.1.2. Model and Data Layer

This section describes the generic data model which was used internally to represent legal documents within the implemented framework. In addition, it introduces the basic notion of the data access layer by explaining, on a concrete example, how the data objects are persisted and mapped into the data storage.

Generic Data Model

This section describes the generic data model which was used to implement the framework to internally represent legal documents. An UML diagram of the data model is shown in Figure 5.2 For the sake of simplicity, concrete attributes, properties, and methods of the classes have been omitted. The main class *LegalDocument* is abstract and divided into three different classes representing the supported document types: laws, judgments, and contracts.

The main attributes are stored as attributes of the classes. Key-Value-Maps support and do not constrain the usage of additional attributes that might even be created at runtime, e.g., the text

analysis engine. Each *LegalDocument* has a class *Metadata* attached, which holds additional information and attributes about a document, e.g., author, created at, etc. The data model can easily be extended to support new document types. Thereby, the new document type needs to be derived from *LegalDocument* and the existing functionalities, including views, search, text analysis engine, etc., can automatically be reused.

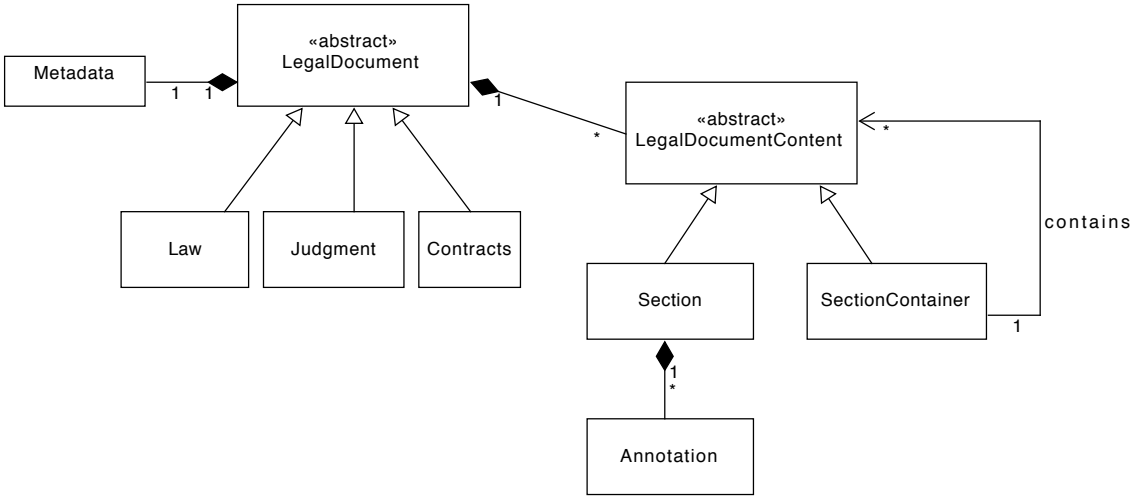


Figure 5.2.: Model for the internal representation of legal documents and annotations.

The content of the *LegalDocument* is implemented as a composite pattern, which was described by Gamma et al. (1994):

LegalDocumentContent: The component is the *LegalDocumentContent*, which is divided into two classes: *SectionContainer* and *Section*.

SectionContainer: The composite is implemented by the *SectionContainer*. This allows to represent the nested structure of German legal documents. For example, the German Civil Code consists of two general parts (i.e., section containers), that are divided into multiple chapters (i.e., section containers), and so forth. This nested structure is part of the law and can be reused during the semantic processing of the legal text. Therefore, this "nestedness" has to be preserved.

Section: The leaves are formed by the *Section* class. This class represents concrete articles of the law. The *Section* also stores the textual information of the articles. They are enumerated, so that the law can be represented accordingly.

Figure 5.3 shows the nested structure of the Geman Civil Code represented as an instantiated composite pattern. The German Civil Code consists of six structural levels, and each of them combines multiple objects, *SectionContainers* and *Sections*. The composite pattern efficiently handles this "nestedness" and adapts to the structure of different laws. It is also open to other document types, such as judgments and contracts. In general, these documents are not nested into so many levels.

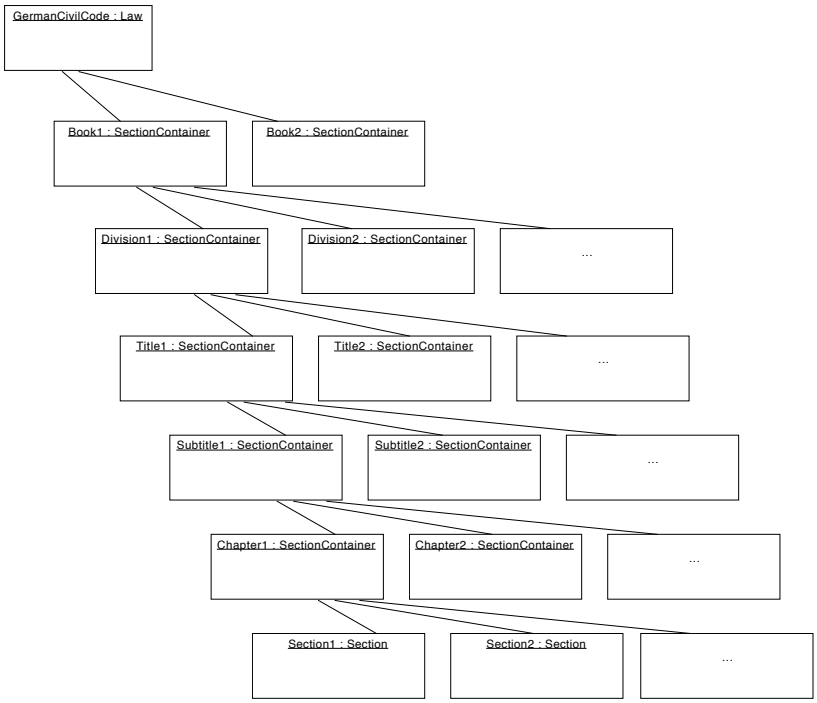


Figure 5.3.: Instantiated object diagram visualizing the nested structure of German laws, e.g., German Civil Code (only two objects per level; remaining objects omitted).

The *Annotation* class is associated with the *Section*. Consequently, each annotation belongs to exactly one *Section*, and if the *Section* is removed or deleted, the annotation is removed as well. A section can have multiple *Annotation* objects. The semantics, and how the annotation objects are created, are described in Section 3.2 and 3.3. The main attributes of the *Annotation* class are:

SectionId References to the *Section* with which an annotation is associated with.

AnnotationType Describes the annotation type of the instantiated annotation.

StartIndex The start offset of the region that is annotated with the referred *Section*.

EndIndex The end offset of the region that is annotated with the referred *Section*.

CoveredText The text that is covered by the region.

The implemented data model has intentionally been kept lean and simple. During the processing, this decreases the effort of handling data objects by resolving references and dependencies. This fosters bulk operations and enables fast serialization and mapping from Java objects into JSON formats, as required by Elasticsearch.

Data Access Layer and Mapping

Based on the data model the different data types are implemented. They are stored in the data storage (Elasticsearch), which is a JSON-based search engine. Therefore, every object that is persisted has to be transformed into a JSON object, as is exemplarily shown in the code listing below.

```
1  package models;
2
3  // imports omitted
4
5  public class Law extends LegalDocument {
6
7      ...
8
9      @Override
10     public void deleteEntity() {
11         super.deleteEntity();
12     }
13
14     @Override
15     protected boolean saveEntityElasticsearch() {
16
17         Map<String, Object> attributes = new HashMap<>();
18
19         if (this.title != null && !this.title.isEmpty()) {
20             attributes.put("Title", this.title);
21         }
22
23         if (this.promulgationDate != null) {
24             attributes.put("PromulgationDate", this.promulgationDate);
25         }
26
27         // mapping of additional attributes omitted
28
29         if (this.isNewEntity()) {
30             String uid = ElasticsearchServer.insert(this.SC_TYPE(), attributes);
31             this.setID(uid);
32         } else {
33             ElasticsearchServer.update(this.SC_TYPE(), this.getID(), attributes);
34         }
35
36         //error handling omitted
37
38         return true;
39     }
40
41     ...
42 }
```

Listing 5.1: The law class overwriting the basic operations, i.e., save and delete.

Listing 5.1 shows two basic operations for the class *Law*, namely deleteEntity() and saveEntityElasticsearch(). Both operations are derived from the superclass *LegalDocument*. The deleteEn-

tity() function only calls the delete operation from the superclass. The saveEntityElasticsearch() creates a map object and subsequently puts every attribute from the class into this object. The listing only shows the mapping for two attributes, namely title and promulgationDate. In the implementation, the remaining objects are handled therein as well. In addition, different processing steps may be required to transform a data type into the expected format, e.g., dates or arrays. An excerpt of the JSON object is shown in listing A.1. Finally, the procedure differentiates whether the document is a new one or one that already exists with its value just being updated (Lines 29 – 34).

This procedure is implemented for every data type that can be persisted in the database. It allows to flexibly adapt the data model and the attributes. As Elasticsearch does not require for a fixed schema definition in advance, the data model can also store attributes that are created and determined at run-time.

5.1.3. Controllers and Request Handling

The controllers handle the HTTP requests. Thereby, different actions in the interface, i.e., the graphical user interface or the REST API, are handled by different controllers. This allows of a clear separation between the responsibilities and functionalities, such as retrieving documents and annotations, searching within the corpus, or semantically analyzing documents.

```
1  package controllers.document;
2
3  // imports omitted
4
5  public class LawController extends BaseDocumentController {
6
7      ...
8
9      public static Result law(String lawId) {
10         Law law = LawDAO.lawForId(lawId);
11         SimpleDateFormat df = new SimpleDateFormat("dd.MM.yyyy");
12
13         JSONObject jsonDoc = new JSONObject();
14
15         jsonDoc.put("id", law.getID());
16         jsonDoc.put("title", law.title);
17         jsonDoc.put("titleShort", law.titleShort());
18         jsonDoc.put("documentType", law.getDocumentType());
19         jsonDoc.put("promulgationDateFormatted", df.format(law.promulgationDate));
20
21         JSONArray sectionContainers = new JSONArray();
22
23         for (SectionContainer ac : law.sectionContainers) {
24             sectionContainers.put(proceedSectionContainer(ac, 0));
25         }
26         jsonDoc.put("sectionContainers", sectionContainers);
27
28         return ok(jsonDoc.toString());
29     }
30
```

```
31      ...
32  }
```

Listing 5.2: Excerpt of the law controller class, showing how requests are handled and how responses are created.

The excerpt of the controller shown in Listing 5.2 handles the request of retrieving a particular law. The law is identified via the lawId parameter, which is part of the HTTP request (Line 9). The controller retrieves the law from the model layer in Line 10. The law variable is a plain java object and mapped into a JSON (Lines 13 – 26) file to be sent back within the HTTP response (Line 28).

The return value of the law function is of the type *Result*, which is a wrapper for the HTTP response. Using the returned object, the Play Framework automatically creates the response object and sends it back to the originator of the request.

As the controllers handle the requests, several controllers are required to handle the different processing and retrieval tasks. In its current implementation, the system contains 19 different controllers. The most important are:

1. **ImportController:** Handles the requests regarding the import and indexing of new documents.
2. **DocumentController:** Handles the requests regarding operations on documents such as retrieval, deletion, and update.
3. **AnnotationController:** Handles the requests regarding operations on annotations such as retrieval, deletion, and update.
4. **ModelController:** Handles the requests regarding operations on semantic models such as retrieval, deletion, and update.
5. **UIMAPipelineController:** Handles the requests regarding operations on processing pipelines regarding UIMA related operations.
6. **AMLPipelineController:** Handles the requests regarding active machine learning operations, such as configuration of pipelines, training and testing of labeled data, and predicting semantic labels.

5.1.4. User Interface and Views

The provision of a user interface to support complex tasks, such as text-intensive search or processing, is known to be non-trivial, and has been studied intensively in computer science and software engineering during the last decades (see Shneiderman et al. 1997; Hearst 2009; Wilson 2011). The implemented system provides a REST API to access from the data storage. This API is consumed by the front-end, which was developed in JavaScript and AngularJS, and uses JSON as exchange format for data objects.

Different design patterns have been established to allows end-users access to the content stored in information systems. We have developed individual views within the prototype to support

each of these different tasks. Thereby, we differentiated between the three basic operations as described by Shneiderman et al. (1997) and Hearst (2009): navigation, exploration, and visualization.

Navigation

Users should have the possibility to access all the imported information by providing proper components. These components include the listing of all the imported documents, navigation menus, and side-bars. Modern frameworks, such as AngularJS, are compatible with software libraries that can easily be reused to rapidly create visually pleasing user interfaces. In our implementation, we reused components from Semantic UI[1] and D3JS[2].

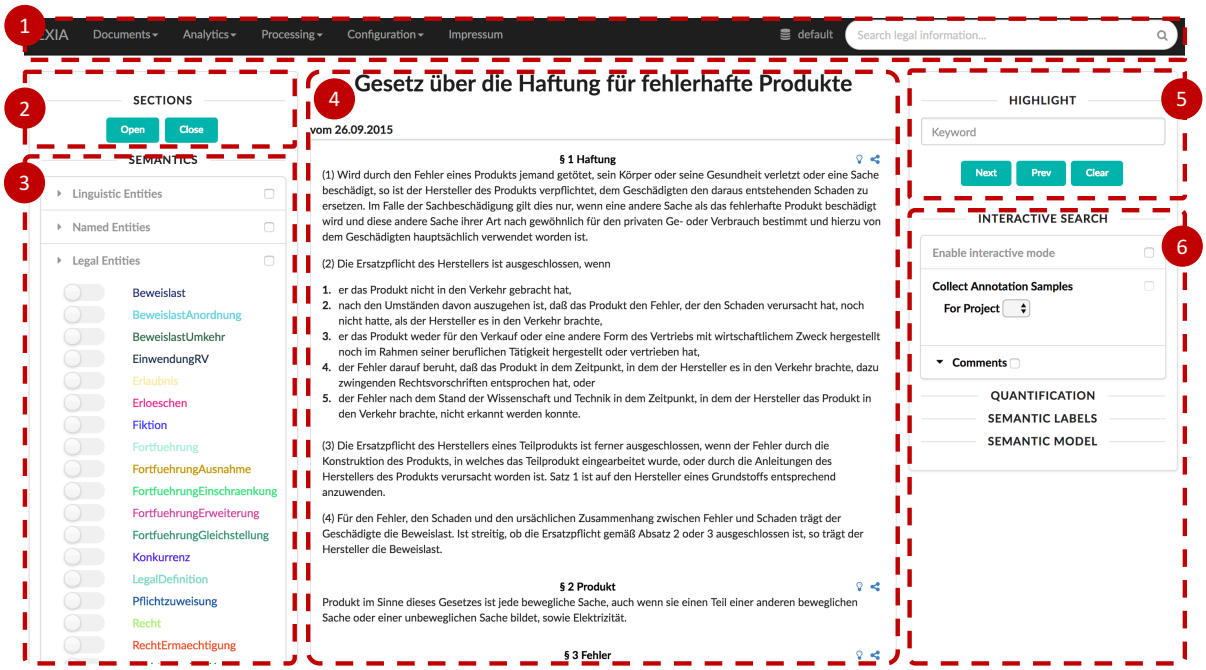


Figure 5.4.: The basic view of a legal document.

Each legal document that is indexed by the data storage can be accessed and its original textual representation is displayed. The view is structured into three different parts: left, middle, and right. The middle part displays the content of the document. On the left and right sides, interactive components are shown that can be used to meta-data of the documents, as well as information about possible annotations. This view is shown in Figure 5.4. It mainly consists of six different main control elements, which are:

1. The **navigation bar** holds a menu to access other views and documents. In addition, it contains a search bar, which allows to perform an advanced full-text search within the document corpus.

[1] https://semantic-ui.com/, accessed on September 3, 2018
[2] https://d3js.org/, accessed on September 3, 2018

2. The **bulk selection** collapses and un-collapses every section contained in the document. This helps to find and read particular sections, and is primarily useful within for larger documents.
3. The **annotation type selection** can be selected and un-selected using check-boxes. Annotations of the selected annotation type are highlighted within the content view.
4. The **content view** shows the title and the textual content of a document. It reflects the document structure, i.e., articles, and highlights annotations, if selected.
5. The **highlighting control** is a convenience feature to efficiently search for a word or phrase within the document. Every occurrence of a given keyword is automatically highlighted.
6. The **meta-data panel** shows additional content of the document, determined metrical information (see Section 3.4.1.4), and separate highlights for annotations.

Exploration

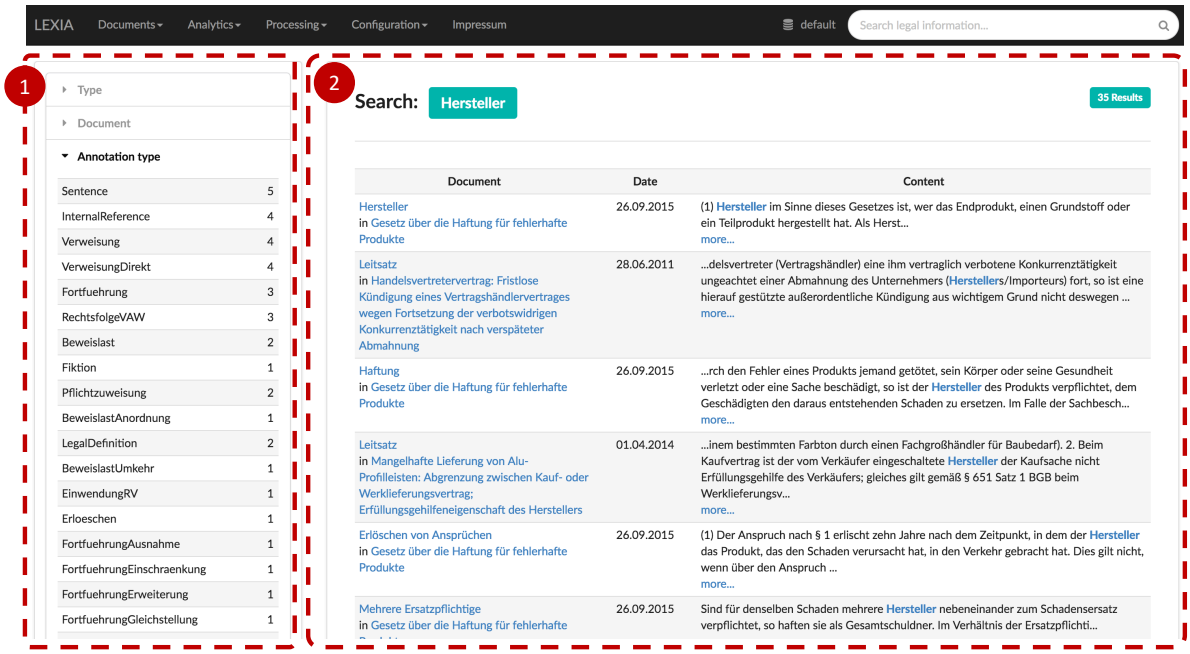


Figure 5.5.: Full-text search results with options for search result refinement.

When it comes to data-intensive tasks in which text is the predominant format, an exploration of the data room can be beneficial. Searching the whole document corpus and refining the search with proper mechanisms, such as facets (Hearst, 2009), can easily be implemented using modern data storages, e.g., ES.

The navigation bar component provides a search bar to start an advanced full-text search, incorporating synonyms and search query processing.

Figure 5.5 shows the result page of a full-text search query for “Hersteller”:

1. **Facets** allow the reduction of the search results according to different dimensions, e.g., document type, document, annotation type, or publishing date.
2. The **result list** shows the list of articles that match with the search query. It is enriched with links to the document and links to the concrete section to fully explore the corpus.

Processing

The management and processing of documents can be performed with a graphical user interface. Especially the processing allows the users to specify various parameters that should be used during the application of the different software components for the linguistic and semantic analysis of legal documents. The user interface supports the specification of many, but not all of the parameters, as this would most probably lead to an information overload in the front-end provides no additional use. Instead, the most promising configuration parameters can only be set statically in the software code.

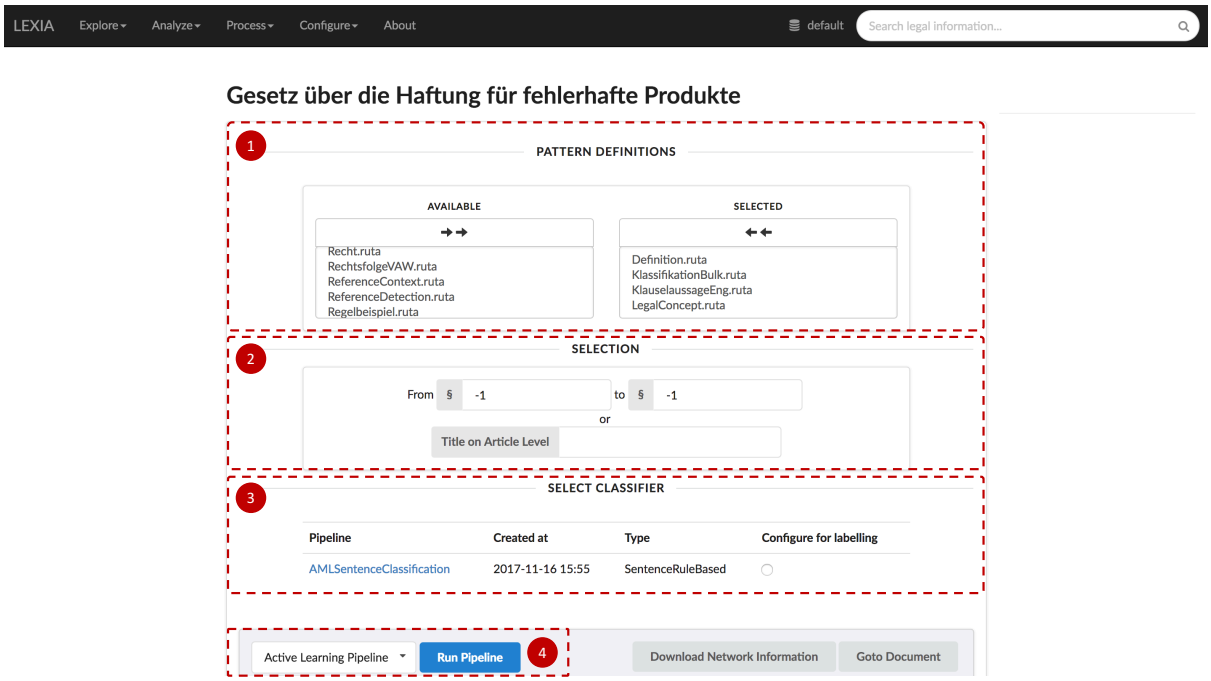


Figure 5.6.: The processing view of a legal document with its four configuration areas.

Figure 5.6 shows the processing page for the document “Gesetz über die Haftung für fehlerhafte Produkte”[3]:

1. **Pattern definitions** are predefined linguistic rules, i.e., Apache Ruta, that can be selected individually. The selected ones are applied within the Apache UIMA pipeline.

[3]Product liability act

2. The **segment selection** offers the possibility for a fine granular specification of sections that are going to be processed. This allows to apply different processing pipelines to a whole document.
3. The **classifier** section lists all the available AML classifiers that are applied in addition to the rule-based pattern definitions (if selected).
4. Finally, different **predefined pipelines** can be selected and be applied to the document with the selected configuration.

5.2. Text Analysis Engine

The text analysis engine is the central component for processing and annotating legal documents. The main components are described in Section 3.5 and the overview has already been discussed there. This section provides more technical details about the integration and concrete implementations.

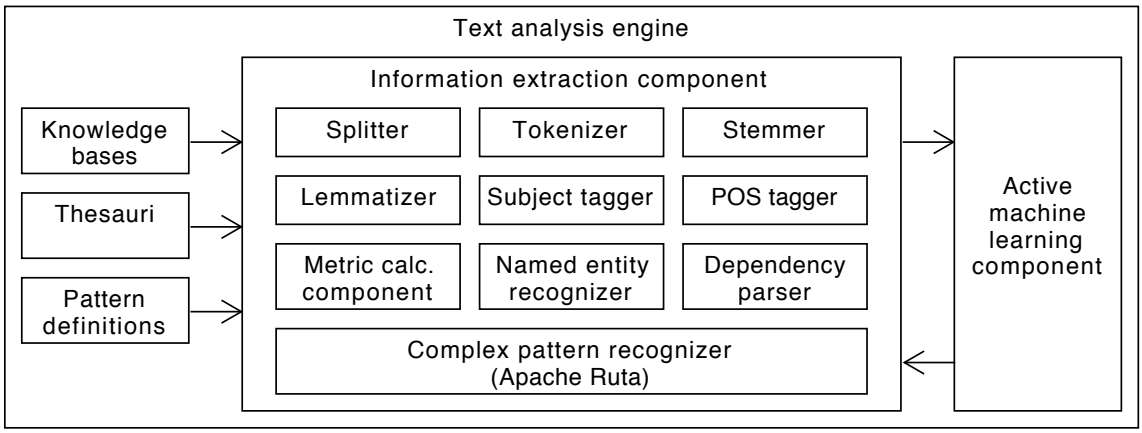


Figure 5.7.: The text analysis engine with its annotators and components.

The next four sections will describe the implementation of the pipeline model using Apache UIMA as baseline (see Section 5.2.1). Subsequently, different notions of annotators are introduced in Section 5.2.2. Finally, the integration of external sources of information, including knowledge bases, thesauri, and pattern definitions, is discussed in Section 5.2.3.

5.2.1. Processing Legal Documents

The processing of the legal documents is an essential part of the implementation. Therefore, the processing pipelines need to be assembled and the sections, or articles, of a legal document are subsequently processed.

```
1  package informationExtraction.pipeline.pipelines;
2
3  // imports omitted
4
5  public class AdvancedPipeline extends Pipeline {
6
7      @Override
8      public AnalysisEngine assemblePipeline(String[] rutaScripts)
9          throws ResourceInitializationException, IOException, InvalidXMLException {
10         return super.assemblePipeline(
11                 rutaScripts,
12                 LawSentenceSegmenter.class,
13                 OpenNlpPosTagger.class,
14                 LawReferenceAnnotator.class,
15                 ArticleHeaderAnnotator.class,
16                 LeitsatzAnnotator.class,
17                 ZitatAnnotator.class,
18                 MoneyValueAnnotator.class,
19                 DateAnnotator.class
20         );
21     }
22
23     @Override
24     public void preArticle(Article article, String text) {
25         ArticleHeaderPattern.setHeader(article.getHeader());
26         super.preArticle(article, text);
27     }
28
29 }
```

Listing 5.3: A pipeline consisting of different software components for linguistic operations.

The creation of a pipeline is shown in Listing 5.3. Therein, a class named AdvancedPipeline is defined. This class extends the superclass Pipeline, which is the base line for all pipelines and supports the integration into Apache UIMA. The main functions of the Pipeline class are shown in Listing 5.4.
The AdvancedPipeline overrides two methods from the superclass:

Lines 7 – 21: AnalysisEngine assemblePipeline (String[ ] rutaScripts): The method specifies a list of annotators that are subsequently within the pipeline. From Lines 12 to 19, the annotators are specified: LawSentenceSegmenter, OpenNlpPosTagger, LawReferenceAnnotator, ArticleHeaderAnnotator, etc. In doing so, the method from the superclass is called (see Listing 5.4). The pipeline also allows the specification of Ruta scripts that are passed as parameters. The parameter is a list of Strings to specify the path to the files containing the Ruta scripts, i.e., rules.

The return value is of the type AnalysisEngine, which is a class from the package org.apache.uima.analysis_engine and therefore natively supported by Apache UIMA.

Lines 23 – 27: void preArticle (Article article, String text): This method is called before an article is processed and allows the specification of information that is particularly relevant for an article. In this case, as the ArticleHeaderAnnotator is applied, the header, i.e., title, of

an article is added to the list of patterns. This ensures that the occurrence of the article header within the article text is annotated accordingly.

As each configuration of a pipeline is derived from the superclass Pipeline the superclass is highly relevant during the analysis phase. Listing 5.4 shows the main components of the Pipeline objects.

```
1  package informationExtraction.pipeline;
2
3  // imports omitted
4
5  public abstract class Pipeline {
6      protected AnalysisEngine pipe;
7      protected JCas jCas;
8
9      ...
10
11     protected abstract AnalysisEngine assemblePipeline(String[] rutaScripts) throws
        ResourceInitializationException, IOException, InvalidXMLException;
12
13     protected static AnalysisEngine assemblePipeline(String[] rutaScripts, Class<?
       extends AnalysisComponent>... c) throws ResourceInitializationException,
       IOException, InvalidXMLException {
14         AnalysisEngineDescription[] d = new AnalysisEngineDescription[c.length];
15
16         for (int i = 0; i < c.length; i++) {
17             d[i] = createEngineDescription(c[i]);
18         }
19
20         AnalysisEngineDescription componentsDesc = createEngineDescription(d);
21         AnalysisEngineDescription rutaDesc;
22         AnalysisEngineDescription completeDesc;
23         if (rutaScripts != null && rutaScripts.length > 0) {
24             rutaDesc = createEngineDescription(UimaUtil.createRutaDescriptions(
       rutaScripts));
25             completeDesc = createEngineDescription(componentsDesc, rutaDesc);
26         } else
27             completeDesc = componentsDesc;
28         return createEngine(completeDesc);
29     }
30
31     public void setup(LegalDocument legalDocument, String[] rutaScripts) throws
       ResourceInitializationException, IOException, InvalidXMLException, CASException
        {
32         pipe = assemblePipeline(legalDocument, rutaScripts);
33         initCas(legalDocument, rutaScripts);
34     }
35
36     protected void initCas(LegalDocument legalDocument, String[] rutaScripts)
       throws ResourceInitializationException, IOException, InvalidXMLException,
       CASException {
37         jCas = UimaUtil.produceJCas(rutaScripts);
38         jCas.setDocumentLanguage(legalDocument.getLanguage());
39     }
40
```

```
41      public PipelineResult process(Article article, String text) throws
        AnalysisEngineProcessException {
42          pipe.process(jCas);
43          return createAnnotationStructures(article);
44      }
45
46      ...
47
48  }
```

Listing 5.4: The pipeline object handling the assemby of software components and the initialization of different components.

The Pipeline class is abstract and specifies several methods. The most important ones for processing legal documents are shown in Listing 5.4 and can be described as follows:

Line 11: AnalysisEngine assemblePipeline (String[ ] rutaScripts): This method is abstract and needs to be provided by every subclass. Within this method, the annotators that need to be applied are specified. In addition, Ruta scripts that are going to be applied to the text are passed as parameters.

Lines 13 – 29: AnalysisEngine assemblePipeline (..., Class<? extends AnalysisComponent>... c): The static implementation of the assemblePipeline method retrieves the specified annotators and creates the AnalysisEngine. Thereby, it iterates over each AnalysisComponent (see lines 16 to 18) and creates AnalysisEngineDescriptors. The Ruta components have to be handled slightly differently, as for each script file, a separate AnalysisEngineDescriptor needs to be created (see Lines 21 to 27). This does not apply if no scripts are provided. Finally, the AnalysisEngine object is created.

Lines 31 – 34: void setup (LegalDocument legalDocument, String[ ] rutaScripts): This method calls the assemblePipeline method from the subclass and initializes the pipe object, which is the central object during the processing phase. At this stage, as every annotator is specified, the Cas object is initialized.

Lines 36 – 39: void initCas (LegalDocument legalDocument, String[ ] rutaScripts): The jCas object is created based on the provided scripts. This is required, since the information for each annotator is specified in the annotator class itself (see Section 5.2.2) but as the scripts may introduce new annotation types, this information has to be injected into the jCas object upfront. The Cas object is passed from annotator to annotator and subsequently enriched with annotations.

Lines 41 – 44: PipelineResult process (Article article, String text): The process method is called for every article of the legal document. The jCas object is already instantiated at this time and the pipeline object is created based on the information provided in the assemblePipeline method. The function as shown does not use the text parameter. Instead, it only calls the process method for the pipe object, which applies all the annotators and creates the annotations in the jCas object.

Finally, a PipelineResult object is created, which extracts the annotations from the jCas

object and maps them into the annotation structure as shown in the data model (see Section 5.1.2).

This section showed the implementation of the pipeline functionality and how different annotators are combined. The next section provides a detailed introduction on how custom annotators are implemented.

5.2.2. Information Extraction Components

The information extraction components are the central parts of the text mining engine. The components are structured into different and independent modules to foster the reuse and the modularity. The next sections will elaborate on the design decisions and exemplify the concrete implementation of a custom annotator.

Reuse of existing annotator components

In Section 5.2.1 and especially in Listing 5.3, the implementation of a pipeline was shown. Accordingly, it became clear that the pipeline manages the interaction between the different annotator components, e.g., sentence splitter, tokenizer, POS–tagger, etc. This includes the exchange of information between those components, e.g., the POS–tagger relies on the token and sentence information that are provided by prior components. This information is stored in the jCas object which is created for Subject of analysis (Sofa). The listings already showed work exclusively on the section level, i.e., the article. Consequently, for each section a new jCas object is created for each section.

As shown in Figure 5.7 and discussed in Section 3.5, a variety of different components have been reused, mainly from freely accessible repositories, such as DKPro[4].

Beside the reuse of components, the pipeline structure is also flexible in that it supports different semantic analysis tasks. The components can be rearranged to extract different semantic types for various purposes. While the main focus of within this research has been set on the analysis of semantic types on a sentence level, it could also be adapted to support the analysis of semantic information on a sub- oder super-sentence level, such as argument mining or detection of conditional phrases.

Implementation of custom annotators for regular expressions

The implementation of a concrete annotator is discussed by ways of an example, namely the extraction of outbound references from legal texts using regular expressions. An excerpt of the class ReferenceAnnotator is shown in Listing 5.5.

```
1 package informationExtraction.semanticAnalysis.uimacomponents.annotator;
2
3 // imports omitted
```

[4]https://dkpro.github.io/, accessed on September 3, 2018

```
4
5  @TypeCapability(
6      inputs = {"de.tudarmstadt.ukp.dkpro.core.api.segmentation.type.Sentence"},
7      outputs = {"informationExtraction.lexiaTypes.InternalReference",
8                 "informationExtraction.lexiaTypes.OutgoingReference"})
9  public class ReferenceAnnotator extends SegmenterBase {
10
11     ...
12
13     @Override
14     protected void process(JCas aJCas, String text, int zoneBegin) throws
       AnalysisEngineProcessException {
15
16         ...
17
18         List<Pattern> outgoingReferencePatterns =
19                     LegalOutgoingReferencePatterns.getExternalReferencePatterns();
20
21         for (Pattern p : outgoingReferencePatterns) {
22             Matcher m = p.matcher(text);
23             while (m.find()) {
24                 if (checkAnnotationAlreadyFound(foundAnnotations, m.start()))
25                     continue;
26
27                 Annotation outgoingReference =
28                             createOutgoingReference(aJCas, m.start(), m.end());
29             }
30         }
31         ...
32     }
33
34     ...
35
36     protected OutgoingReference createOutgoingReference(JCas aJCas, final int
       aBegin, final int aEnd) {
37         int[] span = new int[]{aBegin, aEnd};
38         trim(aJCas.getDocumentText(), span);
39         if (!isEmpty(span[0], span[1]) && isWriteSentence()) {
40             OutgoingReference intRef =
41                                 new OutgoingReference(aJCas, span[0], span[1]);
42             intRef.addToIndexes(aJCas);
43             return intRef;
44         } else {
45             return null;
46         }
47     }
48 }
```

Listing 5.5: A custom annotator accessing regular expressions and applying them to the text of a section.

The ReferenceAnnotator class is annotated to specify the input and output types (see Lines 5 – 8). This information is parsed by the UIMA and defines the interface between the different components. The input type as shown in the listing is of the type Sentence, and valid output

types are InternalReference and OutgoingReference. Other methods that are implemented within the class are omitted, while the process and the createOutgoingReference methods are shown. These perform the main operations during the application of the annotator. The application starts by calling the process function. This is done by the UIMA framework. The process function retrieves the jCas object, the text of the article, and some zoning information. It retrieves the pattern definitions as regular expressions from the class LegalOutgoingReferencePatterns in Line 17.

The pattern definitions are applied to the text via standard Java functionality of regular expressions (see Lines 20 to 29). Once a pattern matches, a new annotation object outgoingReference is created (see Line 26 and 27). Thereby, the method createOutgoingReference is called, which creates a new annotation object based on the position information of the matched text (Lines 40 and 41) and adds it to the jCas (Line 42). The annotation object of the type OutgoingReference contains the exact position of a matched string in terms of character position (offset). Based on this information, it can automatically retrieve the covered text information by applying a string operation. This operation is provided by the UIMA framework and is implemented as straight-forward substring method: text.substring(getBegin(), getEnd()) in Line 128 of the class org.apache.uima.jcas.tcas.Annotation[5].

5.2.3. External Resources: Pattern Definitions, Thesauri, and Dictionaries

As some of the annotators require access to additional resources, such as pattern definitions, or information from a dictionary, these dependencies are implemented as well.

Pattern Definitions

The pattern definitions are either stored as Ruta scripts or as a collection of regular expression. The latter ones are stored either in a file or in static Java classes, such as those shown in Listing 5.5 Line 19. If the expressions are stored in a separate file, it can be manipulated via a file system. This is an advantage, but can also be considered as a disadvantage, since it is more vulnerable to corruption.

Ruta scripts are stored in separate files. This is necessary due to the overall architecture of Apache Ruta. The script files are created within the user interface and can be applied while processing a legal document (see Figure 5.6).

Thesauri and Dictionaries

Thesauri and dictionaries can be integrated into the overall system to enable a clear distinction between the logic of the applied rules the lexical information, such as vocabularies for synonyms. This is a common design principle in text analysis engines. Within the GATE framework, these dictionaries are called gazetteers. The thesauri and dictionaries usually contain a list of entities, such as cities, organizations, days of a week, etc. However, they can also contain more and

[5] https://uima.apache.org/, accessed on September 3, 2018

semantic information such as dependencies between terms, e.g., synonyms, or hypernyms. These lists can be accessed via wrapper classes written in Java. Therefore, the information is available to the annotators in the processing phase.

5.3. Active Machine Learning Component

The AML component is part of the text analysis engine and allows the creation of annotations for phrases and sentences based on machine learning functionality. The main functionality was discussed in Section 3.5. This section provides more technical details about its integration and the implementation of its main parts.

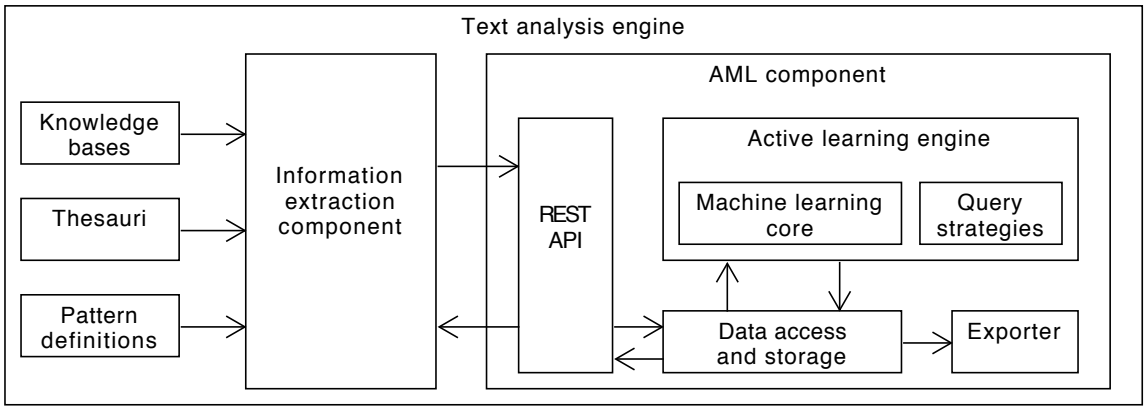


Figure 5.8.: The text analysis engine with its annotators and components.

The next five sections describe the implementation of the pipeline model using Apache Spark as the baseline. For instance, the interaction with the information extraction component is discussed in Section 5.3.1. The main parts are the configuration and training of the models (see Section 5.3.2), the persisting of models (see Section 5.3.3), and the prediction of instances (see Section 5.3.4). As query strategies are essential for AML, their role and a concrete implementation are discussed in Section 5.3.5.

5.3.1. Interaction Between Information Extraction and AML Component

This section describes the interaction between a user, i.e., domain expert, the information extraction component, and the active machine learning component. The overall process is subdivided into four different phases:

1. Configuration
2. Training
3. Evaluation

4. Prediction

An overview of the process is shown as a sequence diagram in Figure 5.9. It consists of three different interacting entities, namely the domain expert, the information extraction, and the AML component.

Configuration: The configuration phase is required to set up the initial parameters that are used in the machine learning task. It starts with creating an AML pipeline. The pipeline is parametrized with the labels that are available during the classification task. This input has to be provided by the domain expert, as the AML is a subtype of supervised machine learning. The remaining parameters for the AML have to be configured, such as the size of the seed set, the amount of learning rounds, the query strategy, etc.

Finally, a legal document can be selected with content that will be classified using the defined parameters. The document is split into its Sofas by the information extraction component. These do not necessarily have to be sentences, but could also be phrases or tokens. In this research project, the main focus was however set on the classification of sentences.

Training: The training phase starts by initializing the AML procedures by the domain expert, who is provided with proper user interfaces. This triggers the information extraction component, which in turn triggers the AML component. Since the AML component runs as a stand-alone service, it is accessed via a REST interface. To start the training of the model, the configuration is loaded from the store and the parameters for learning pipelines are set.

The training process starts by labeling a first, randomly selected set of instances, i.e., a seed set. Based on this set of instances, the AML trains the model for the first time and calculates the next training instances using its query strategy. After this first training, the learning rounds are executed (red function in Figure 5.9). Within this function, sentences, i.e., instances, are subsequently delivered to the AML, which in turn returns the next instances, that need to be labeled. The repetition of learning rounds is halted once the stopping criterion is reached.

Evaluation: The evaluation phase is optionally performed after the training. Therein a test set is passed to the AML component, which predicts a label for each of the instances. The predictions are returned as result and compared to a pre-defined gold standard. Based on this comparison the confusion matrix can be determined and common evaluation metrics, such as precision and recall, are calculated.

In the experiment to evaluate the overall performance (see Section 6.4), the evaluation phase is performed after each learning round. This allows the assessment of the learning behavior and the impact of query strategies.

Prediction: The prediction phase is a straight-forward machine learning task. The information extraction component passes instances to the AML module. In doing so, labels are predicted based on the trained model. No interaction with the domain expert is required during this phase, i.e., no training is performed.

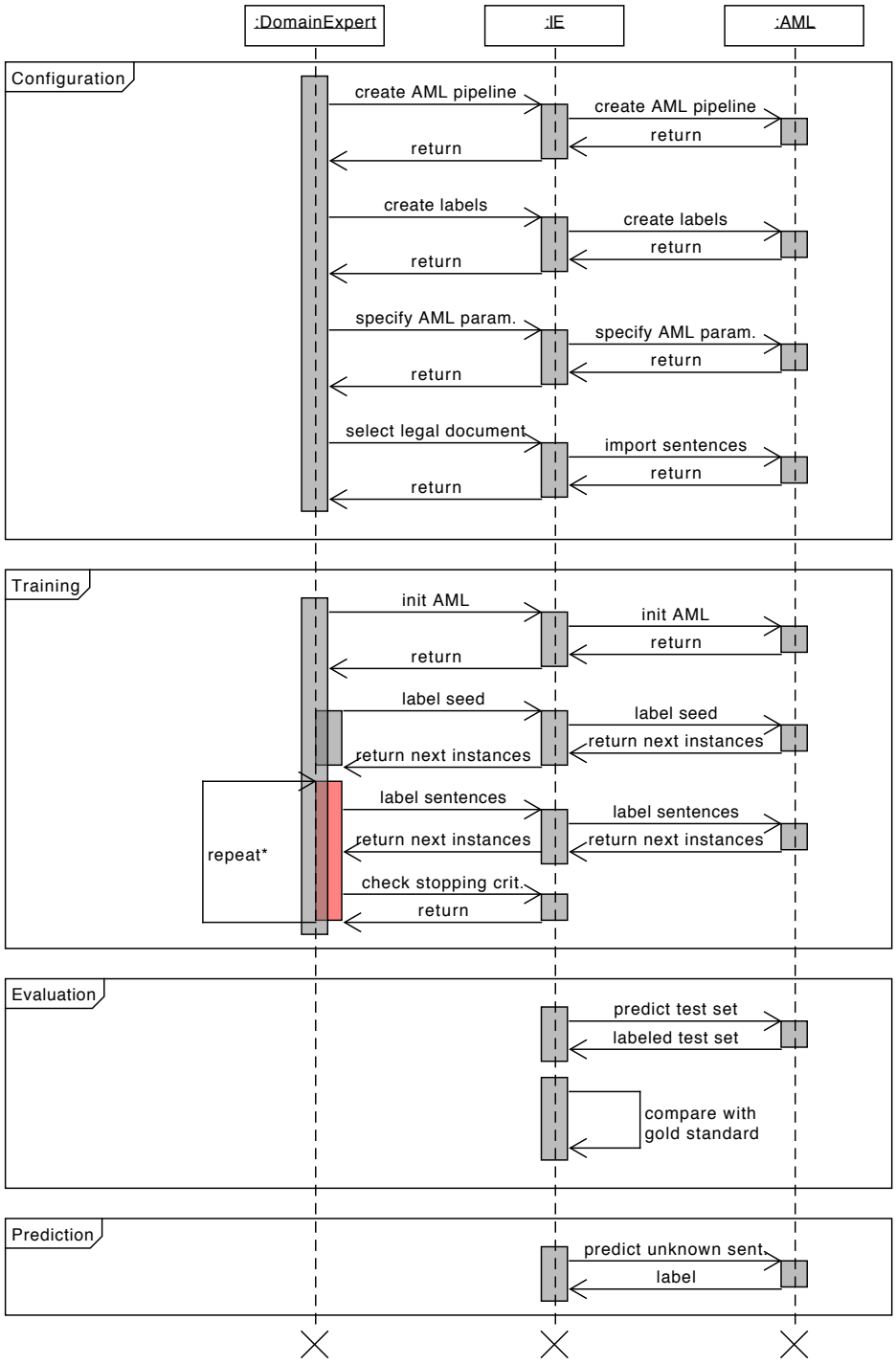


Figure 5.9.: Configuration, training, evaluation, and prediction of instances, such as sentence types, with AML.

5.3.2. Configuration and Training of Models

The configuration of the models is carried out by the domain expert who provides the required information via the user interface, i.e., REST API. The proper method in the AML component is shown in Listing 5.6. The listing shows how different parameters are subsequently extracted from the configuration as provided in the HTTP request (see Line 1). In this process, five main phases are performed and finally the pipeline configuration is saved:

1. **Instantiation of active learning pipeline** ins Line 4 and 5
2. **Select classifier** in Lines 8 and 9
3. **Select query strategy** from Lines 12 to 14
4. **Select seed set size** in Lines 17 and 18
5. **Learning round configuration** from Lines 21 to 26

```
1  JsonNode configurations = request.get("alPipelineConfigurations");
2
3  // load pipeline configurations
4  ALPipeline pipeline = ALPipeline.getALPipelineCurrentStatus(pipelineName);
5  ConfigurationPipeline configurationPipeline = new ConfigurationPipeline();
6
7  // set classifier as specified in http request
8  String classifier = configurations.get("classifier").asText();
9  configurationPipeline.setClassifier(pipeline, classifier);
10
11 // set query strategy as specified in http request
12 String queryStrategy = configurations.get("queryStrategy").asText();
13 pipeline.getAlPipelineConfigurations()
14         .setQueryStrategy(QueryStrategyFactory.getQueryStrategy(queryStrategy));
15
16 // set seed set size as specified in http request
17 int seedSet = configurations.get("seedSet").asInt();
18 configurationPipeline.setSeedSet(pipeline, seedSet);
19
20 // set learning round parameters as specified in http request
21 int minRound = configurations.get("stoppingCriteria").get("minRound").asInt();
22 int maxRound = configurations.get("stoppingCriteria").get("maxRound").asInt();
23 int querySize = configurations.get("querySize").asInt();
24
25 configurationPipeline.setStoppingCriteria(pipeline, minRound, maxRound);
26 configurationPipeline.setQuerySize(pipeline, querySize);
27
28 pipeline.save();
```

Listing 5.6: Configuration based on information provided by the REST interface.

The classifier is selected as string from the user, but internally, it is handled differently. In Line 9 of Listing 5.6, the classifier of the pipeline is set. This is done choosing one of the pre-defined classification models. Currently, three different classifiers are implemented, namely, MultinomialNaiveBayes, LogisticRegression, and MultilayerPerceptron.

The Listing 5.7 illustrates the prototypical implementation of the DefaultLogisticRegression based on the existing implementation of LogisticRegression[6]. Thereby, the mllib parameters are set to pre-defined default values.

```
1  package processing.classifier;
2
3  // imports omitted
4
5  public class DefaultLogisticRegression implements IClassifier,
       IProbabilisticClassifier {
6
7      public static LogisticRegression get() {
8
9          return new LogisticRegression()
10                 .setMaxIter(10)
11                 .setElasticNetParam(0.8)
12                 .setRegParam(0.001);
13     }
14 }
```

Listing 5.7: Implementation of Logistic Regression with exemplary parameters.

Once the configuration of the active learning is stored and the classifier is selected, the training phase is initiated. Thereby, a TrainingPipeline object is created, which retrieves all the required information from the previously specified ALPipeline object. The relevant code parts are shown in Listing 5.8.

The method preparePipeline sets the internal parameters and fields according to the passed ALPipeline object (see Lines 13 – 18). This includes the stages, i.e., learning rounds, the training and test data set parameters, and the parameters regarding the unlabeled data.

The TrainingPipeline class has a method called label, which iterates over the provided training records and adds them to the set of labeled instances (see Lines 20 – 33). Once the instances are added, the pool of training instances is cleared and the pipeline is saved (see Lines 31 and 32).

```
1  package processing.pipeline;
2
3  // imports omitted
4
5  public class TrainingPipeline {
6
7      private Dataset<Row> trainingData = null;
8      private Dataset<Row> unlabeledData = null;
9      private PipelineModel model = null;
10
11     // other methods omitted
12
13     public void preparePipeline(ALPipeline pipeline) {
14         setStages(pipeline);
15         createTrainingAndTestSet(pipeline);
```

[6] https://spark.apache.org/docs/2.2.0/mllib-linear-methods.html#logistic-regression, accessed on September 3, 2018

```
16          setUnlabeledData(pipeline);
17          pipeline.save();
18      }
19
20      public void label(Map<String, String> idLabelMap, ALPipeline pipeline) {
21          for (Map.Entry entry : idLabelMap.entrySet()) {
22              Integer label = getLabelKey(pipeline.getLabelMap(), (String) entry.
    getValue());
23              for (Row row : pipeline.getTrainingsData().getDataToLabelNext()) {
24
25                  if (row.get(0).equals(entry.getKey())) {
26                      pipeline.getTrainingsData().getLabeledData().add(RowFactory.
    create(row.get(0), (double) label, row.get(1)));
27                      break;
28                  }
29              }
30          }
31          pipeline.getTrainingsData().getDataToLabelNext().clear();
32          pipeline.save();
33      }
34  }
```

Listing 5.8: The training pipeline is built upon the ALPipeline object and encapsulates the required methods for the labeling of instances.

The TrainingPipeline separates the responsibilities and functionalities of the ALPipeline object from the remaining tasks, such as configuring and predicting. Once a model has been configured and was trained, it can be used to predict the semantic type, i.e., label, of a new instance.

5.3.3. Persistence of Models

Once a model is trained, or even after each learning round, the model is persisted. This has two main advantages:

1. Decoupling of learning rounds
2. Load trained model for prediction

Since the model is safely stored after a learning round, the different rounds can be carried out over a large period of time. They need not necessarily be performed immediately one after another. The learning process can be interrupted and resumed after a learning round, as soon as the domain expert has free resources, e.g., time, available. Additionally, this allows to reuse the already trained model for prediction tasks. Persisting the trained model is common and the main use case to perform a prediction on unlabeled data.

```
1  package processing.pipeline;
2
3  // imports omitted
4
5  public class TrainingPipeline {
6
7      // other methods omitted
```

```
8
9       public void saveModel(ALPipeline pipeline) {
10          FileUtils.deleteDirectory(new File("sparkModels/" + pipeline.getName()));
11          model.save("sparkModels/" + pipeline.getName());
12          pipeline.setModelPath("sparkModels/" + pipeline.getName());
13          pipeline.save();
14      }
15  }
```

Listing 5.9: The model of a classification pipeline is persisted.

Each TrainingPipeline can be persisted and the main method is shown in Listing 5.9. Thereby, two different saving operations need to be differentiated, saving the configuration into a document-oriented database (see Line 25) and persisting the trained model to the file system (see Lines 28 – 33). Storing the trained model on the file system is a common procedure, as the model can be very large and might not be handled efficiently by databases. Additionally, these models are binaries and treated as Binary Large Objects (BLOBs), which are per se hard to handle for databases.

5.3.4. Predicting of Instances

Just like the TrainingPipeline, the prediction task is also encapsulated within a separate class, namely PredictionPipeline. The main methods are shown in Listing 5.10. Thereby, the task is orchestrated within the method executePrediction (see Lines 13 – 19). The model is loaded from the storage (Line 14 and Section 5.3.3) and the instances are transformed into the required internal structure (Line 15).

The actual prediction of the labels for instances is performed with the applyModel method. Calling the transform operation for the model (see Line 26) applies the specified configuration. The results can be collected in a subsequent loop for each instance.

```
1  package processing.pipeline;
2
3  // imports omitted
4
5  public class PredictionPipeline {
6
7      private List<Row> data = new ArrayList();
8      private Dataset<Row> dataset = null;
9      private PipelineModel model = null;
10
11     // other methods omitted
12
13     public ArrayNode executePrediction(ALPipeline pipeline) {
14         loadModel(pipeline.getModelPath());
15         createDataFramePrediction(pipeline);
16         ArrayNode results = applyModel();
17         cleanUp();
18         return results;
19     }
20
```

```
21      public void loadModel(String modelPath) {
22          this.model = PipelineModel.load(modelPath);
23      }
24
25      public ArrayNode applyModel() {
26          // perform prediction operations on the dataset
27          Iterator predictions = this.model.transform(dataset).cache().toJSON().
        collectAsList().iterator();
28
29          // collect information, ie labels, about instances and return them
30          ArrayNode result = Json.newArray();
31          while (predictions.hasNext()) {
32              result.add(Json.parse(predictions.next().toString()));
33          }
34          return result;
35      }
36
37  }
```

Listing 5.10: The application of a model and classifier using the Apache Spark framework.

Finally, Apache Spark expects a method call, which allows to free allocated resources. This is recommended for very resource-intensive tasks, e.g., the labeling of large document sets.

5.3.5. Query Strategies

Query strategies, as described in Section 3.5.5, are used to determine those training instances which are most informative and therefore increase the learning performance of classifiers. To illustrate their mode of operation, an example is shown in Listing 5.11.

```
1  package processing.queryStrategy.uncertaintySampling;
2
3  // imports omitted
4
5  public class MostUncertainEntropyStrategy
6      extends AbstractEntropy
7      implements IQueryStrategy {
8
9      @Override
10     public List<Row> getInstancesToLabelNext(List<Dataset<Row>> predictionsList) {
11
12         List<Row> uncertainRows = new LinkedList<>();
13         Dataset<Row> predictions = predictionsList.get(0).cache();
14
15         // analyse each available predicted instance
16         for (Row row : predictions.collectAsList()) {
17
18             int probability = row.fieldIndex("probability");
19             int prediction = row.fieldIndex("prediction");
20             int path = row.fieldIndex("path");
21             int text = row.fieldIndex("text");
22             Vector probVector = (Vector) row.get(probability);
23             Double selectedLabel = row.getDouble(prediction);
```

```
24              double confidence = probVector.apply(selectedLabel.intValue());
25
26              // determine entropy measure for each instances
27              double entropy = calculateEntropy(probVector.toArray());
28
29              uncertainRows.add(RowFactory.create(
30                                      row.get(path),
31                                      selectedLabel,
32                                      confidence,
33                                      entropy,
34                                      row.get(text)));
35          }
36          return uncertainRows;
37      }
38  }
```

Listing 5.11: Implementation of the query strategy "most uncertain entropy".

The class MostUncertainEntropyStrategy is derived from the base class AbstractEntropy, which provides methods to determine the entropy measures of a given probability vector, as used in Line 24. The class consists of one method, namely getInstancesToLabelNext. The method obtains a list of all the predicted instances and is called at the beginning of a learning round. During the method, each instance is analyzed and for each instance, a row record is created (see Lines 26 – 31) that contains information about the instance and the determined entropy measure (see Lines 17 – 24). Finally, the list of records is returned and the calling method can extract those instances, which are most informative according to this entropy criterion.

5.4. Implementation of the Model-based Reasoning Framework

The concept and the design of the model-based reasoning system were already described in Section 4. Figure 5.10 provides an overview of the system architecture with the three extensions on the user interface, modeling component, and the reasoning engine. The integration of these components is discussed within this section. In addition, main implementations and architectural and design decisions are described.

As shown in Figure 5.10, the modeling component is divided into two sub-components, namely for the semantic modeling, including the representation and handling of types, attributes, and relations; and for the executable modeling, focusing on the specification of derived attributes as MxL expressions. The user interface has been extended so that both parts of a model can be formalized accordingly. In addition, the knowledge acquisition component enables the application of models by providing facts and by instantiating the models (see also Section 4.3.2). The explanation capability, as provided by the execution and reasoning engine, is integrated into the knowledge acquisition component, as main parts focus on providing an explanation for the derivation of an attribute.

The next five sections describe the implementation of the main parts of the model-based reasoning framework. First, the interaction between domain experts, the modeling component, and the reasoning engine is introduced as a sequence diagram in Section 5.4.1. The modeling

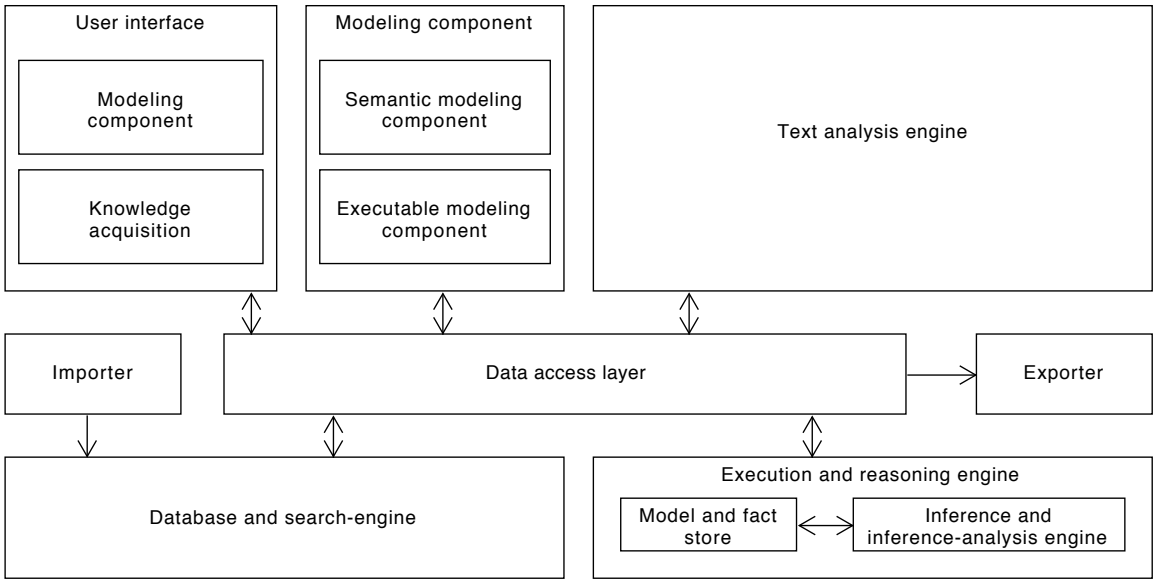


Figure 5.10.: System architecture focusing on the components that enable model-based reasoning.

component, including a discussion on the mapping of the data model with the reasoning engine, is provided in Section 5.4.2. Based on these considerations, the access of the reasoning engine is described in Section 5.4.3. The main methods for storing instances and facts from the knowledge acquisition component are investigated in Section 5.4.4. Finally, the process of retrieving the explanation for an derived attribute is described in Section 5.4.5.

5.4.1. Domain Experts, Modeling Components, and a Reasoning Engine

This section describes the interaction between a user, i.e., domain expert, the information extraction component, and the active machine learning component. The overall process is subdivided into three different phases:

1. Modeling
2. Application
3. Explanation

An overview of the process is shown as a sequence diagram in Figure 5.11. It consists of four different interacting agents, respectively components, namely the domain expert, the modeling component, the knowledge acquisition component, and the model storage.

Modeling: The process starts by formalizing the semantic model. Thereby, the domain expert creates types, attributes, and relations. The requests from the front-end and user interface are processed by the modeling component. Minor consistency checks are already performed

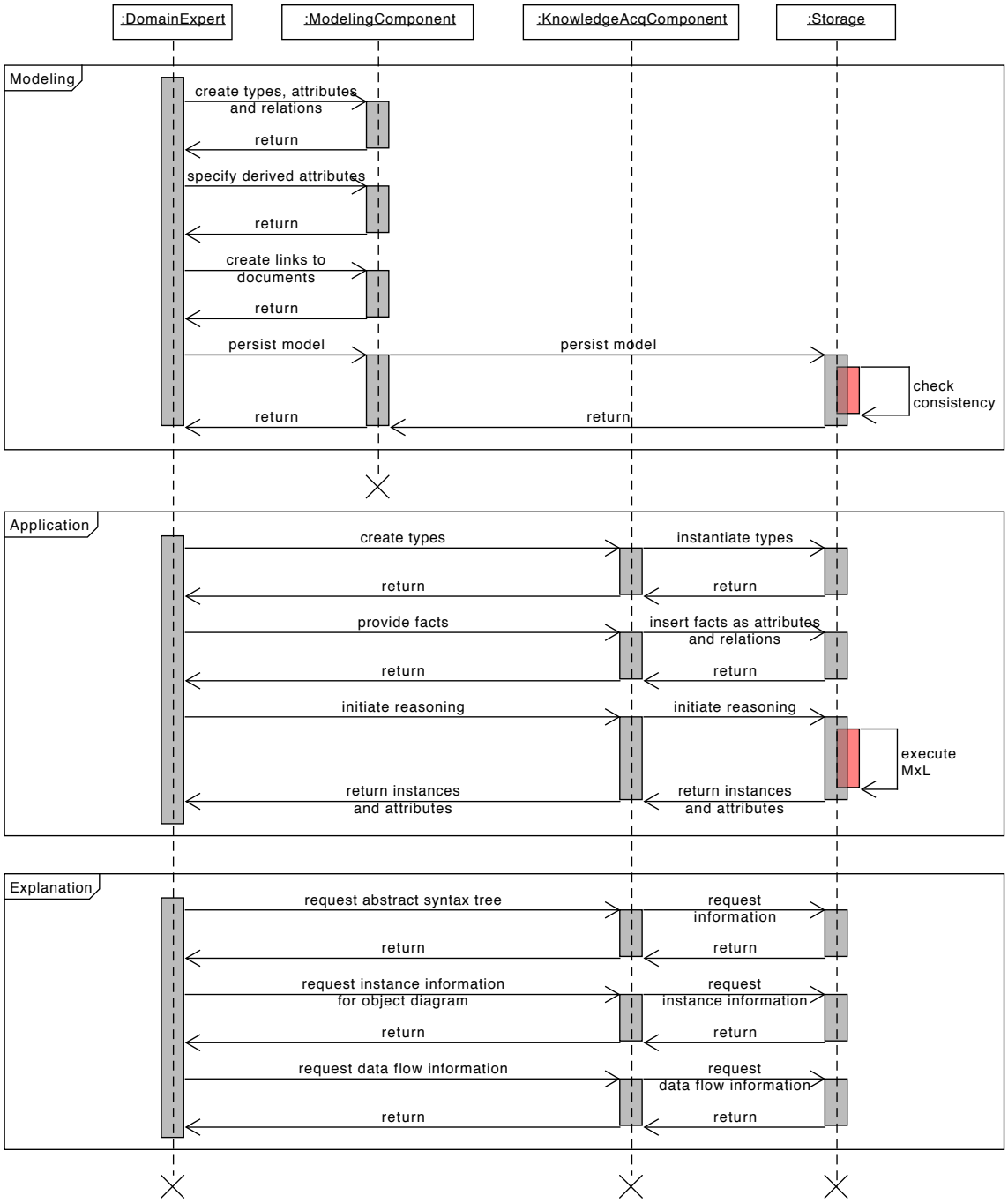


Figure 5.11.: Interaction between domain experts and the software services and tools.

at this stage, either by the user interface or by the modeling component. After all the

static model elements are defined, the derived attributes need to be specified. Based on the types, attributes, and relations, the derived attributes define the executable semantics.

The model elements are optionally linked to the text, the interpretation of which led to the creation of the model element. These links are associations between documents, sections, or annotations.

Once every model element is defined, the user initiates persisting the model into the model store. The modeling component maps the data elements to the structure as required by the model store and wraps them in proper REST calls. The storage retrieves this data, checks its consistency, and — in case of success — persistently stores the model elements.

After the modeling phase, the modeling component is no longer required during the process.

Application: The application phase is initiated by the end-user via the knowledge acquisition component. He creates instances of the types and inserts the facts, i.e., evidence, into the instantiated model. The knowledge acquisition component wraps this information into the required format and passes it forward to the model storage, where the facts are stored along with the model information (see also Section 4.3.2).

The user can initiate the reasoning by requesting the value of a derived attribute from the storage. The reasoning engine within the model storage executes the MxL expressions and determines the concrete values of an attribute. This deductive reasoning process illustrates how new knowledge is derived from existing facts using the predefined rules, i.e., expressions.

Explanation: Once an end-user has created the model and received the information about concrete reasoning on provided facts he has the possibility to retrieve explanations about concrete derivations. These three models include an overview of the instances and their attributes as object diagrams (see Section 4.4.1), information about the underlying decision structure as formalized in the AST (see Section 4.4.2 and 4.4.3), and a data flow inspection of the dependencies and influences of attributes (see Section 4.4.4).

For each of the three information requests, the interaction sequence follows the same path: The information request is initiated by the user and received by the knowledge acquisition component. The component in turn processes the request and forwards it to the model storage and reasoning engine.

5.4.2. Modeling and Knowledge Acquisition

In order to handle the communication and the transfer of information between the user, the modeling environment, and the reasoning engine, the data as entered in the user interface needs to be wrapped and mapped. The mapping of the model elements is essential to access the functionality as provided by the model storage and reasoning engine. Both components are used as external services and accessed via the REST API. The components have been developed independently from this particular research but have been adapted accordingly (see Reschenhofer et al. 2016).

Semantic model element		**Model and fact store**
Model	$\mapsto$	Workspace
Type	$\mapsto$	EntityType
Atomic attribute	$\mapsto$	AttributeDefinition
Relation	$\mapsto$	AttributeDefinition (as Link)
Derived attribute	$\mapsto$	DerivedAttributeDefinition

Table 5.1.: Mapping to meta-model based IS (see Neubert 2012).

The system is a meta-model-based information system. The data model of the system is highly flexible and enables modeling tasks. The mapping of model elements is shown in Table 5.1. The table illustrates the elements of the semantic model-based reasoning approach with the information system that serves as a model and fact store.

The graphical representation of Table 5.1 is provided in Figure 5.12. The left side of the figure shows the elements of the semantic model store. A model has many different types and each type can have multiple attributes. For sake of simplicity, the relations and derived attributes are not explicitly modeled, but subsumed as attributes. The meta-model of the information system that was adapted for this purpose consists of workspaces and EntityTypes, which refer to models and types respectively. The AttributeDefinition object reflects the definition of attributes

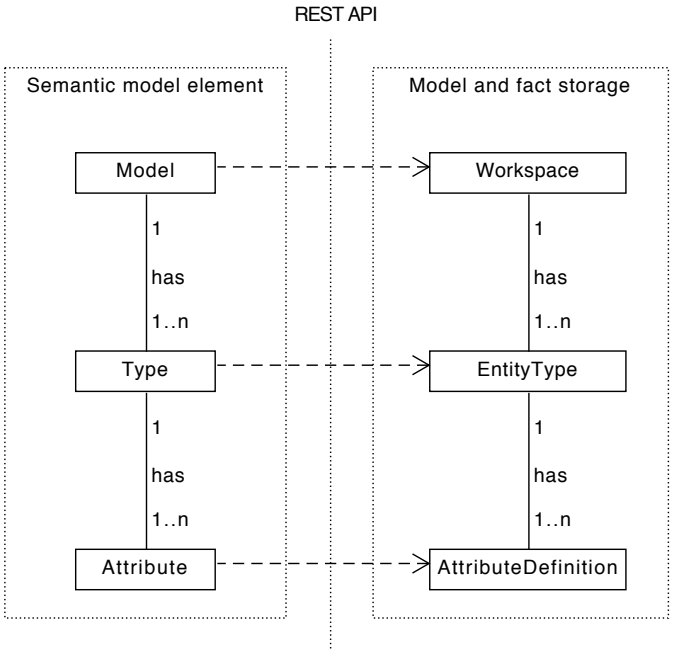


Figure 5.12.: Simplified data model to illustrate the mapping (see Oppmann 2016).

for an EntityType. The AttributeDefinition allows for a specification of different data types, such as Boolean, string, MxL, but also for links, i.e., relations, which refer to other EntityTypes.

5.4.3. Accessing the Model and Fact Store

The actual access model and fact store are carried out using a visitor pattern. The visitor pattern is a well-known behavioral pattern in the field of software engineering and is considered as an efficient implementation to perform operations on potentially nested data structures. As the user-created model potentially consists of multiple elements, the visitor pattern allows to efficiently iterate over each of these elements and to perform an operation. Listing 5.12 illustrates the concrete implementation that was used in the implemented prototype.

```
1  public void convertModel (JsonNode node, Model graphModel)  {
2
3      JointJSGraph graph = mapper.readValue(node.toString(), JointJSGraph.class);
4
5      //sort the cell list: first the types are created, thenrelations
6      Collections.sort(graph.getCells(), new CellBeforeLinkComparator());
7      graph.setGraphModel(graphModel);
8
9      //visitor for persisting entity types with attributes and links
10     graph.accept(new MetaModelVisitor());
11
12     //visitor for persisting the derived Attributes Definition (MXL)
13     graph.accept(new DerivedAttributesVisitor());
14
15     //save the graph because the IS ids have been added
16     graphModel.setJsonModelDefinition(Json.toJson(graph).toString());
17     graphModel.saveEntity();
18
19 }
```

Listing 5.12: Visitor pattern to save the nested structure of a semantic model.

The method convertModel retrieves the information about the model from the front-end visualization component. The information as modeled by the user is stored in the variable node and mapped into the graph object in Line 3. In Line 6, the model components are sorted, since the storage needs to get the information about types first, then attributes, and finally links. The visitor pattern execution is initiated in Lines 10 and 13. This initiates an iteration over all elements of the graph and performs an action according to the type. Basically, every element is persisted in the information system via the REST API. First, the basic elements are persisted, and then the derived attributes. Finally, the model is also stored locally to keep the information for the visualization in the front-end (lines 16 and 17).

5.4.4. Knowledge Acquisition Component

The knowledge acquisition component retrieves data from the user interface and inserts the data into the fact storage. In the front-end the model information is rendered, so that the user has the

possibility to insert and update data in a form-based dialog (see Section 4.3.3). The front-end already prevents invalid input types. For example, if an attribute is of the type Boolean, the front-end offers the user the selection of "true" and "false".

Once the user saves his input, the back-end functionality is triggered, which receives the data in a JSON format. The JSON object is parsed, mapped to the data format of the fact storage, and persistently saved. The main methods are show in Listing 5.13.

```
1  public void saveInstancesInIS(JsonNode node, Model graphModel) {
2
3      JointJSGraph graph = mapper.readValue(node.toString(), JointJSGraph.class);
4
5      // the order matters: types, attributes, relations, mxl
6      Collections.sort(graph.getCells(), new LinkBeforeCellComparator());
7      // persist in fact store
8      graph.accept(new PartialUpdateVisitor(graphModel));
9
10     graphModel.setJsonModelDefinition(Json.toJson(graph).toString());
11     graphModel.saveEntity(); // store meta-information in local database
12 }
13
14 public void deleteAllInstanceData (JsonNode node, Model graphModel) {
15
16     JointJSGraph graph = mapper.readValue(node.toString(), JointJSGraph.class);
17
18     //delete all data facts
19     graph.accept(new DeleteInstancesVisitor());
20
21     graphModel.setJsonModelDefinition(Json.toJson(graph).toString());
22     graphModel.saveEntity(); // store meta-information in local database
23 }
```

Listing 5.13: Persisting and deleting instances and facts.

Listing 5.13 shows two main methods for the interaction between an end-user and the fact store. The saveInstancesInIS retrieves the JSON node object from the front-end and maps it into a graph representation (see Line 4). The elements of the graph are sorted as the fact store requires it. First, types and attributes are persisted, then relations, and finally derived attributes. This ensures consistency during the insertion of data, as the fact store automatically checks the dependencies and therefore, needs atomic attributes first.

In Line 8, the visitor pattern is used to persist all the model elements in the fact store. Each model element and its value is visited and stored in the remote fact storage. Lines 10 and 11 are executed to persist the current status of the model in the local database. It is important to note that no information about facts is stored locally, but only meta-information about the model and the model elements, which are required to map the model elements accordingly.

The deleteAllInstanceData method performs the analogous procedure for the deletion of information. The main difference is the visitor operation that is called: DeleteInstanceVisitor (Line 18).

5.4.5. Explanation Component

The explanation component consists of four different and complementary parts:

- Instance and Fact View(see Section 4.4.1)
- Abstract Syntax Tree for Dependency Analysis (see Section 4.4.2)
- Explanation Dialog Component (see Section 4.4.3)
- Data Information Flow Inspection (see Section 4.4.4)

This section illustrates how the explanation for a given attribute is retrieved from the reasoning engine. Thereby, the reasoning engine is accessed via the REST API.

```
1  // ModelController.java
2  public static Result validateMXLExpression() {
3
4    JsonNode data = request().body().asJson();
5    String entityId = data.get("entityId").asText();
6    String mxlExpression = data.get("mxlExpression").asText();
7
8    String result = ISController.evaluateMXLExpression(entityId, mxlExpression);
9
10   return ok(result);
11 }
12
13 // ISController.java
14 public static String evaluateMXLExpression(String entityId, String mxlExpression) {
15
16   // ... remain code omitted
17
18   String url = "entities/" + entityId + "/mxlValidation";
19
20   ObjectNode node = Json.newObject();
21   node.put("expression", mxlExpression);
22
23   HttpURLConnection connection = connectionForPostRequest(url);
24
25   // ... remain code omitted
26 }
```

Listing 5.14: Retrieving explanations from the reasoning engine.

The main procedures for the retrieval is shown in Listing 5.14. The main methods are validateMXLExpression and evaluateMXLExpression. The validateMXLExpression is called from the front-end upon user request. The MxL expression and the data that identifies an attribute are loaded (Lines 4 – 6). Using this information, the reasoning engine is accessed (see Lines 18 – 23). Thereby, the entity is unambiguously identified, using its URL with the mxlValidation extension (Line 18). The information that is collected during the execution of the expression is aggregated with in a JSON object, which is forwarded to the front-end (Line 10). The front-end parses it and creates the visualization, i.e., tree structure.

5.5. Summary

This chapter described the implementation of the concepts as introduced in Chapter 3 and Chapter 4. The main focus was set on the providing technical details and design decisions that have been made during the software technical implementation. The chapter is subdivided into four different parts, which follow the main implementation areas of the prototype:

1. Collaborative Data Science Environment
2. Text Analysis Engine
3. Active Machine Learning Component
4. Model-based Reasoning Framework

The basic principle for the collaborative data science environment was the implementation to a web application. The application follows a Model-View-Controller pattern, which allows to pursue a modular system architecture and to separate different software components with regard to their function. A generic data model was used to support the adaption to different document types, such as laws, judgments, and contracts. Additionally, the mapping between the database objects and the objects as required by the Java framework, and the JavaScript front-end was introduced. Main parts of the front-end were shown in detail to illustrate how end-users can access the information and functionality.

A main part of the back-end implementation is the so-called text analysis engine, which contains the functions and methods to semantically process and annotate legal documents. The documents are processed in a Pipes & Filters architecture, which allows the subsequent application of different software components. These components can either be existing software modules that are freely accessible, or newly implemented components, which provide a highly specified functionality of information extraction and annotation. Along this processing pipeline, information is exchanged via the JCAS object.

The active machine learning components extends the text analysis engine by additional machine learning functionality to classify semantic types within legal documents. The component is implemented as a stand-alone service, but coupled to the information extraction component via a standardized interface. Using this interface, the main phases of the AML can be parametrized and initiated, namely configuration, training, evaluation, and prediction. Trained models are persisted so that they can be reused without resource intensive training. This persistence requires an additional store.

To not only support the textual analysis of legal documents, but also the computational legal reasoning on interpreted legal norms, a model-based reasoning framework was implemented. It combines the functionality of a meta-model based-information system with the requirements for model-based reasoning on interpreted legal norms. The implementation includes a user interface and the adoption of the existing system, which is solely accessed using its REST API. The chapter describes the three phases, namely modeling, application, and explanation, and specifies the main methods that are required to achieve the overall functionality. Important design decisions and code listings illustrate the implementation.

Based on the concept, the design, and the implementation, the next chapter describes the evaluation of the different parts of the system and discusses its performance and limitations.

CHAPTER 6

Evaluation and Assessment

6.1. Evaluation Approach

In the previous chapters, the main contribution of this work was introduced. Initially, Chapter 3 described the concepts and basic framework for the textual analysis of legal documents, including rule-based and active machine learning-based components. In the following Chapter 4, a logical framework, namely model-based reasoning, was introduced to extend classical knowledge engineering by ontologies, with the possibility to define logical and arithmetical expressions. These models and the formalizations are the results from the interpretation of statutory texts. Chapter 5 discussed various aspects of the software technology aspects, as well as the most central implementations in detail.

For design science in information systems research, Hevner et al. (2004) mentioned different evaluation strategies, including and twelve concrete methods to evaluate an implemented prototype or research artifact. Based on the evaluation framework in Hevner et al. (2004) and Peffers et al. (2007), the prototype is assessed in different aspects of performance, including case studies (see Eisenhardt 1989), standard information retrieval metrics (see Salton 1989), and functionality studies (see Gregor 2006).

The evaluation of the full-stack implementation was performed on three aspects, namely on the usage within an interdisciplinary data science project to support editorial processes on tax law documents, the information extraction performance, and on the formalization of a decision structure from the German tenancy law.
The aspects are covered by four evaluation set-ups:

1. **Support of editorial processes**
2. **Text analysis: rule-based**

3. **Text analysis: active machine learning-based**
4. **Formalizing termination notice periods of Germany's tenancy law**

The first three evaluation approaches are mainly empirical and assess the applicability of the framework. They also serve as a base line for discussions about the limitations and challenges for the integration of the framework for practical use cases (see Section 6.2). The fourth evaluation assesses the adaptability of the formalization of model-based reasoning. While, evaluation set-ups two and three focus on the textual and linguistic semantics of the German tenancy law (see Sections 6.3 and 6.4), the formalization, including the semantic and executable model (termination notice periods), is discussed in this fourth evaluation task (see Section 6.5).

6.2. Case Study: Analysis of Fiscal Court Cases to Support Editorial Processes

Background and Motivation

In 2016, a joint research project was performed with the German industry partner, namely Datev eG[1]. Datev eG is "a software company and IT service provider for tax consultants, auditors, and lawyers as well as their clients"[2]. As of December 31^{st}, 2016, the Datev eG had more than 7,005 employees and a turnover of approx. 928 million Euros. Datev eG is an association, i.e., cooperative, with more than 40,000 members.

In Germany, the company is known for their IT services that are offered to its members and customers. Its service portfolio includes a legal information database, called LEXInform[3]. This database provides access to information on various topics that are relevant for legal practitioners and scientists. The main focus, however, is in the legal domain of tax law.

The documents are not only produced by Datev eG, but also official documents from legislation, and jurisdiction. These documents must be pre-processed, so they can be indexed accordingly. Thereby, different information has to be extracted from the document, and meta-data has to be attached.

The task, which should be performed using the prototype developed in this thesis, was to support this manual extraction process with rule-based information extraction methods. The focus was set on the extraction of a particular piece of information from tax law cases. The information that needed to be extracted was the Year of Dispute (YoD)[4]. The YoD is particularly relevant for the tax law domain, as it indicates which version of the German tax law was applied in a given case. As the tax law changes regularly, including minor and larger revisions, this information is essential during the litigation process. However, editorial staff of official courts does not assign this information.

[1] https://www.datev.com, accessed on September 3, 2018
[2] see official web page Datev eG (2017)
[3] https://www.datev.de/dnlexom/client/app/index.html, accessed on September 3, 2018
[4] dt. Streitjahr

Objective

Given a large set of tax law cases, functionalities were implemented and the research prototype was adapted, so that the software could support during the extraction of the YoD within any case document. The YoD must not necessarily be one specific year, but it can also be a set of years, i.e., a timespan. This set must not necessarily be subsequent.

The extraction task was reduced to an annotation problem, as most of the cases contain this information within their textual information. Case files which do not contain the information about the YoD have not been considered during the evaluation. The objective was to automatize — as far as possible — the extraction of this particular information.

Roles

The task was performed by two researchers from the academic research group. Part of this small research group were two legal data scientists, one with a technical software engineering background, and the other with a legal practice background. The industry partner had three persons involved: two from the "Department for Portals and Collaboration", i.e., legal data scientists, and one from the "Department for Content - Taxes and Law", i.e., a legal practitioner.

Data and Import

The document corpus provided by the Datev eG consisted of more than 130,000 different documents related to the German tax law. The documents covered a time span of almost 100 years. The oldest documents within the corpus were from 1919, whereas the latest document in the corpus was published in July 2016. The corpus was fully digitized (no OCR required) and available in XML, while each document was represented by a single file.

The corpus consists of more than 40 different types, such as judgments[5], articles[6], and laws[7], etc. The corpus is a selection of the documents stored in the Datev eg legal information database LexInform. Within this task, the corpus was restricted to judgments. This led to the overall amount of 47,359 different case documents. The distribution over time is shown in Figure 6.1.

Processing and Approach

The process as introduced in Section 3.1, was used for the analysis task:

Import The first task was to import and index the documents within the prototype. This could be performed straight-forward, by adapting the import structure and proper methods (see Section 3.4.1.1).

[5] dt. Urteile
[6] dt. Aufsätze
[7] dt. Gesetze

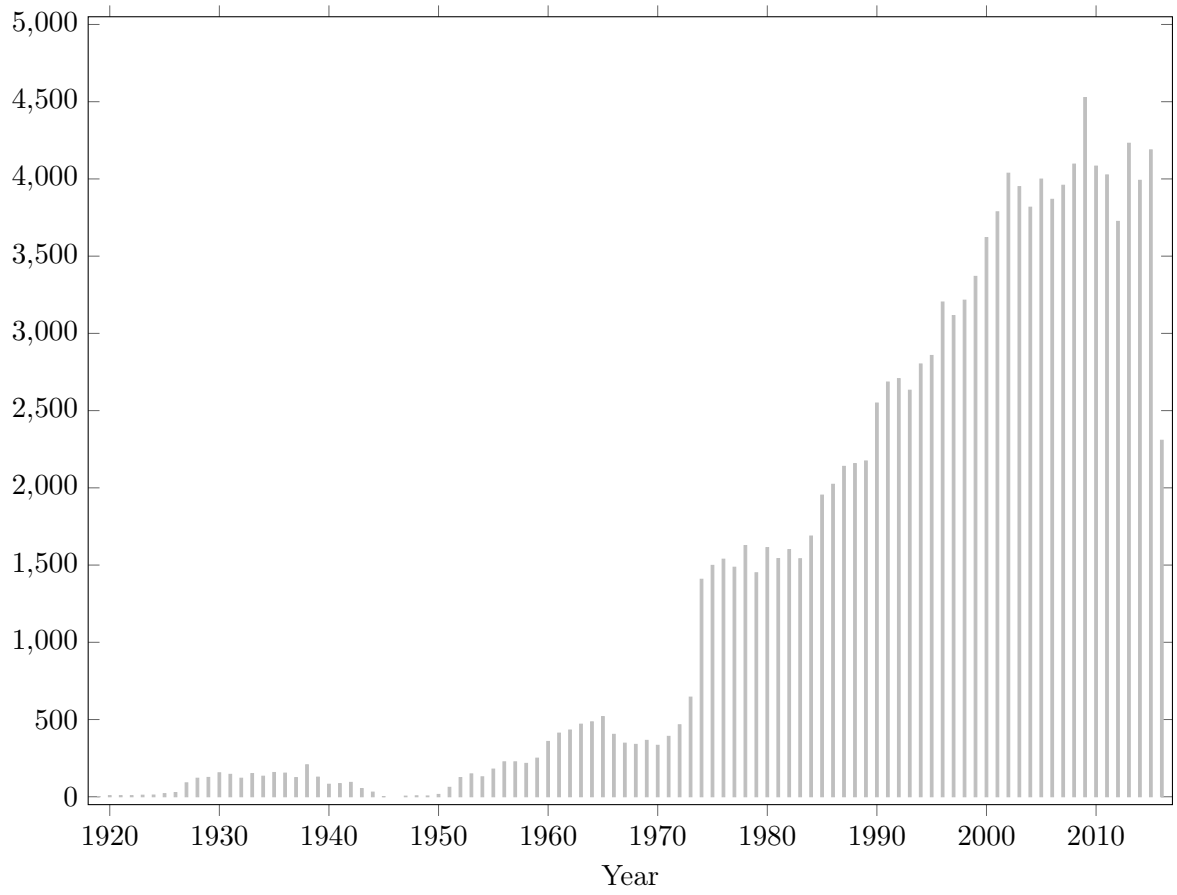


Figure 6.1.: Judgments in the tax law document corpus provided by Datev eG (Σ 47,359 docs).

Analyze Once the documents had been imported and the index structure was updated, the analysis was performed. Thereby, a zoning strategy was used to constrain the search space in which the information about YoD occurs. The YoD is described within the section "statement of facts"[8], so the algorithm only analyzes this particular section of a case.
This restriction of the search space decreased the false positive rate and therefore contributed to the accuracy of the overall approach, which is described in the following steps:

1. Constrain search space to section "statement of facts"
2. Determine dates within the text
   - Differentiate between specific dates, e.g., "21.10.2010" or "21. September 2010", dates referring to a whole year, e.g., "2010", and a timespan "2000 bis 2009".
3. Determine indicating sentences, such as "Antragssätze"
   - If those sentences contain whole years or timespans, mark those as year of disputes

[8] dt. Tatbestand

4. Determine contexts that allow conclusions about the YoD, based on particular linguistic patterns expressed in Apache Ruta.
   - Pre-Indicators: Linguistic features, i.e., tokens, words, patterns, indicating that the following date is likely to be the year of dispute.
     Examples: "auf die im Streitjahr 2006 zugeflossenen Erstattungszinsen", "den Einkommensteuerbescheid 2006 vom 11.12.2007", "die Kindergeldfestsetzung für den Zeitraum von Oktober 2003 bis Dezember 2004 und von Januar 2006 bis Juni 2006", etc.
   - Post-Indicators: Linguistic features, i.e., tokens, words, patterns, indicating that the date mentioned before is likely to be the year of dispute.
     Examples: "Erwerbsunfähigkeitsrente im Jahre 2005 (Streitjahr)"
   - Clamp-Indicators: Linguistic features, i.e. tokens, words, patterns, indicating that the date mentioned between two features is likely to be the year of dispute.
     Examples: "ab Januar 2008 Kindergeld zu bewilligen", "Bescheid für 1997 und 1998 über Einkommensteuer"
5. Extract and annotate the literal indicating the YoD.
6. Evaluate and assess the performance of the analysis.

The analysis phase was performed iteratively. This iteration included steps 2 – 6.

Application The YoD have been extracted using the methodology as shown above. The resulting annotations were persisted and highlighted in the text view of the document.

Extraction Performance

During the evaluation phase, 100 different case documents were randomly selected and verified. The results are shown in Table. Note that a case can have multiple years of dispute.

		Predicted Outcome	
		YoD	**No YoD**
Actual Outcome	**YoD**	186	21
	No YoD	11	-

Table 6.1.: Quality assessment of YoD extraction in 100 randomly selected cases.

Based on the evaluation and the confusion matrix (see Table 6.1) precision values of 0.94 and a recall of 0.90 could be determined. Hence, the F_1 measure is approximately 0.92 for the extraction.

Discussion and Critical Reflection

This result could be improved even further by providing a more comprehensive set of rules and vocabulary. For many cases, this approach is sufficient and reliable to extract the YoD, but there

$$\begin{aligned} \textbf{F1} &= \frac{2 * 186}{372 + 32} = \frac{372}{404} \approx \mathbf{0.92} \\ \textbf{Precision} &= \frac{186}{186 + 11} = \frac{186}{197} \approx \mathbf{0.94} \\ \textbf{Recall} &= \frac{186}{186 + 21} = \frac{186}{207} \approx \mathbf{0.90} \end{aligned}$$

Table 6.2.: Quality metrics calculated from the confusion matrix of Table 6.1.

are boundary cases that do not allow to fully extract the desired information on the linguistic level with high confidence. Within the corpus, cases exist, in which many different facts are described over several years, so that the algorithm using the rule-based annealing approach easily makes mistakes and provides wrong or insufficient results. Against this background, the approach can support extraction processes by an editorial staff, rather than making autonomous decisions.

The main challenge posed by the extraction of the year of dispute from case documents is the linguistic variety in which this information is lexically and syntactically encoded; especially the huge time span and different types of German courts that are publishing the documents.

The process used was well-suited for this interdisciplinary task and the activities import, analysis, and applications have subsequently been performed. This task showed the necessity of an interdisciplinary and interacting project group when it comes up to the analysis of legal documents with the objective to analyze documents to extract a specific piece of information.

6.3. Performance Evaluation: Classifying Legal Norms with Rule-based Information Extraction

Motivation

Due to several reasons, the classification of norm sentences is an attractive research area within legal informatics. First of all, assigning semantic types allows the differentiation of a norm's function and thus supports subsequent interpretation. Second, the semantic type can be embedded into search and exploration tasks in legal information databases and can therefore support search and exploration tasks within legal documents. And finally, it can be considered as an additional step to extract references and dependencies between legal norms.

With regard to the state-of-the-art in information extraction, rule-based approaches are still predominantly used in the industry Chiticariu et al. (2013). Although rule-based approaches are known to require tedious manual labor, they are comparably easy to comprehend and to maintain. In addition, it is much easier to incorporate domain knowledge from domain experts, which widely exists in the legal domain.

Classification of legal norms can be addressed from different perspectives: Classification with

regard to the content and the topics addressed by a norm, or with regard to its functional aspects. For this evaluation, a classification regarding functional aspects was chosen. Functional classes of norms address the role of a norm within a given legal statute, such as obligations, permissions, or prohibitions. These classes were derived bottom-up and harmonized within a taxonomy.

Objective

The objective of this evaluation was to assess the performance of rule-based information extraction to classify legal norms from German statutory text regarding their semantic type, i.e., functional class. The publicly available data was used for this purpose, namely German federal laws from the domain of civil law.

The rule-based information extraction approach was chosen in order to set a base line for the classification of legal norms and to compare it against a more advanced approach by using active machine learning. The rule-based approach was performed on a taxonomy, which was created by a domain expert, i.e., legal expert, and was derived bottom-up. The rules have been created in a standard language for pattern-based information extraction, namely Apache Ruta. Four iterations have been performed, in which the rules have subsequently been refined to increase the performance of the classification task.

Data

For the first attempt of a functional classification, nine different semantic types of legal norms have been identified: duty, indemnity, permission, prohibition, objection, continuation, consequence, definition, and reference. Table 6.3 gives an overview of the different classes, including a description for each type. Thereby, the description focuses on the primary function of a legal norm. It is clear that long norms may have multiple functions, such as duty, reference, and definitions. However, the classification only took into account the main functional aspect. For the subsequent classification task using active machine learning, the taxonomy was extended even further, taking into account a more differentiated perspective on the functional categorization of legal norms in German civil law documents (see Section 6.4).

Table 6.4 shows concrete examples for each semantic type. The examples are extracted from the Title 5 “Lease, usufructuary lease” - Subtitle 1 “General provisions for leases” from the German Civil Code[9]. The table lists the German sentences and their official translations and provides an illustrative approach to the abstract definition of possible semantic types. It also shows that norms could potentially fall in to two (or more) categories, such as the example for the consequence, which could also be interpreted as a reference type. However, the classification focused on the primary function of a sentence, which would be the consequence. Obviously, the demarcation is rather fuzzy and requires clear definitions and illustrative examples.

In order to prepare a dataset, which serves as gold standard, for the norm classification experiment, one legal expert assigned a semantic type to every sentence of the tenancy law section in the German civil code (§535 - §595) published on March 1st, 2017. The taxonomy has been

[9] https://www.gesetze-im-internet.de/englisch_bgb/englisch_bgb.html, accessed on September 3, 2018

Table 6.3.: Semantic types of norms in German civil law statutes.

	Semantic Type	**Description**
I	Duty	The primary function of a duty is to stipulate actions, inactions or states.
II	Indemnity	The primary function of an indemnity is to clarify that, resp. under which conditions a duty does not exist.
III	Permission	The primary function of a permission is to authorize actions, inactions or states.
IV	Prohibition	The primary function of a prohibition is to forbid or disallow actions, inactions or states.
V	Objection	The primary function of an objection is to define that, resp. under which circumstances an existent claim may not be asserted.
VI	Continuation	The primary function of a continuation is to extend or limit the scope of application of a precedent legal statement.
VII	Consequence	The primary function of a consequence is to stipulate legal effects, without ordering or allowing character as far as the legal consequence part is concerned.
VIII	Definition	The primary function of a definition is to describe and clarify the meaning of a term within the law.
IX	Reference	The primary function of a reference is to cite another norm with the aim of total or partial application transfer or non-application.

designed so that a norm can be mapped on exactly one label. In case of conflicts, e.g., when a could be assigned to multiple labels, it was assigned to the class to which it primarily belongs. For example, if a statement is a right and a legal definition, the legal expert decided which was the primary role of the norm regarding it functional type and usage within interpretation.

The overall task resulted in a gold standard consisting of 601 labeled sentences with nine different labels, as shown in Table 6.5. The table shows that the support of the different semantic types varies throughout the dataset. This variation is not unusual for real datasets, but causes challenges during the experiment and especially in machine learning approaches, where separate strategies exist to deal with unbalanced datasets.

Experimental Setup

The approach was done considering the reference process as described in Section 3.1. The German Civil Code was imported into the application, then split with regard to its sections and sentences. Based on the sentence information, four iterations of writing rules and applying them to the indexed document were performed. The rules were written directly in the implemented system, which it also persisted, and applied it to the text.

After each iteration, the performance of the rules has been evaluated. Based on the results of the evaluation the rules have been adapted and improved. This adaption was carried out by a domain expert, who thoroughly analyzed the errors of the rules, and by a legal data scientist

Table 6.4.: Examples of semantic types of norms from the German Civil Code.

	Semantic Type	**Example (German)**	**Example (English)**
I	Duty	Der Mieter ist verpflichtet, dem Vermieter die vereinbarte Miete zu entrichten.	The lessee is obliged to pay the lessor the agreed rent.
II	Indemnity	Veränderungen oder Verschlechterungen der Mietsache, die durch den vertragsgemäßen Gebrauch herbeigeführt werden, hat der Mieter nicht zu vertreten.	The lessee is not responsible for modifications to or deterioration of the leased property brought about by use in conformity with the contract.
III	Permission	Die Vertragsparteien können eine andere Anlageform vereinbaren.	The parties to the contract may agree on another form of investment.
IV	Prohibition	Ferner kann der Vermieter sich nicht auf eine Vereinbarung berufen, nach der das Mietverhältnis zum Nachteil des Mieters auflösend bedingt ist.	In addition, the lessor may not invoke an agreement by which the lease is subject to a condition subsequent to the disadvantage of the lessee.
V	Objection	Eine zum Nachteil des Mieters abweichende Vereinbarung ist unwirksam.	A deviating agreement to the disadvantage of the lessee is ineffective.
VI	Continuation	Dies gilt nicht, wenn der Mieter gekündigt hat.	This does not apply if the lessee has given notice of termination.
VII	Consequence	Kennt der Mieter bei Vertragsschluss den Mangel der Mietsache, so stehen ihm die Rechte aus den §§ 536 und 536a nicht zu.	If the lessee knows of the defect when entering into the agreement, then he does not have the rights under sections 536 and 536a.
VIII	Definition	Ein Mietspiegel ist eine Übersicht über die ortsübliche Vergleichsmiete, soweit die Übersicht von der Gemeinde oder von Interessenvertretern der Vermieter und der Mieter gemeinsam erstellt oder anerkannt worden ist.	A list of representative rents is a table showing the reference rent customary in the locality, if the table has been jointly produced or recognized by the municipality or by representatives of lessors and lessees.
IX	Reference	§ 551 Abs. 3 und 4 gilt entsprechend.	Section 551 (3) and (4) apply with the necessary modifications.

Table 6.5.: Manually labeled dataset consisting of sentences extracted from the German tenancy law.

Semantic Type		**Occurrence**	**rel. Occurrence**
I	Duty	117	19%
II	Indemnity	8	1%
III	Permission	148	25%
IV	Prohibition	18	3%
V	Objection	98	16%
VI	Continuation	21	3%
VII	Consequence	117	19%
VIII	Definition	18	3%
IX	Reference	56	9%
		Σ 601	100%

(the author of this thesis), who implemented the changes and their implications on the rules. In total, four iterations were performed. The evaluation was done using standard metrics for the quantitative assessment of information extraction tasks, which is described in the next section in detail.

Evaluation

The objective of this experiment was to evaluate the degree to which legal norms in legal texts can be classified with regard to the functional categorization as shown above. Thereby three main issues have been analyzed:

1. The accuracy to which legal norms can be classified using rule-based information extraction.
2. The continuous improvement using multiple iterations and providing the performance measures as feedback to the domain expert after each iteration.
3. The trade-off between precision and recall in classifying legal norms with regard to the effort that is made to create and maintain the rules for the classification.

The quantitative performance has been assessed by evaluating the results of the rules in four iterations. To compare the performance standard, the following evaluation metrics were used:

1. Precision
2. Recall
3. F_1

The evaluation as shown in Table 6.6, depicts the nine semantic types of legal norms within the table. For each semantic type, the precision, recall, and F_1 are determined within each of

Table 6.6.: Four iterations of rule-based norm classification in German tenancy law.

Semantic Type			**Iterations**			
			I	**II**	**III**	**IV**
I	**Duty**	Precision	**0.673**	0.658	0.630	0.634
		Recall	0.497	0.626	0.839	**0.839**
		F1	0.571	0.642	0.720	**0.722**
II	**Indemnity**	Precision	0.194	0.194	**0.715**	0.714
		Recall	0.375	0.375	0.385	**0.385**
		F1	0.255	0.255	0.500	**0.500**
III	**Permission**	Precision	**0.886**	0.854	0.822	0.822
		Recall	0.531	0.530	0.831	**0.831**
		F1	0.664	0.654	0.827	**0.827**
IV	**Prohibition**	Precision	0.327	0.286	0.857	**0.857**
		Recall	**0.500**	0.100	0.316	0.316
		F1	0.395	0.148	0.462	**0.462**
V	**Objection**	Precision	0.895	**1.000**	0.990	0.983
		Recall	0.298	0.048	0.893	**0.922**
		F1	0.447	0.091	0.939	**0.951**
VI	**Continuation**	Precision	0.947	0.947	0.947	**0.950**
		Recall	0.514	0.545	0.600	**0.633**
		F1	0.667	0.692	0.735	**0.760**
VII	**Consequence**	Precision	0.406	0.242	0.824	**0.832**
		Recall	0.211	0.238	0.748	**0.748**
		F1	0.278	0.240	0.784	**0.788**
VIII	**Definition**	Precision	0.146	0.127	0.157	**0.295**
		Recall	0.250	0.400	0.381	**0.520**
		F1	0.185	0.193	0.222	**0.377**
IX	**Reference**	Precision	0.783	0.833	0.833	**0.833**
		Recall	0.771	**0.873**	0.696	0.696
		F1	0.777	**0.853**	0.759	0.759
Arithmetic Mean (weighted)		Precision	0.697	0.674	0.798	**0.803**
		Recall	0.435	0.427	0.771	**0.781**
		F1	0.518	0.465	0.773	**0.782**

the four iterations. The overall weighted arithmetic mean improved from 0.518 (after the first iteration) to 0.782 (after the fourth iteration).

The results show the general performance of the classifier improves, but by incorporating the feedback of the domain expert, it also occurred that the F_1 score of certain classes significantly dropped. For example, this happened for the classes "Prohibition" after the first iteration (F_1: from 0.395 to 0.148) or for "Objection" (F_1: from 0.447 to 0.091). A possible explanation can be given by inspecting the strategy how rules are induced and created. If the domain expert realizes that a rule is too broad and applies for too many norms, which diminishes the precision, the scope of the rule is narrowed. In a subsequent application, the rule may then be too narrow and misses many right classifications. This is a pattern that we constantly observed during the experiment.

Critical Reflection

The evaluation used established methods in information extraction to classify the semantic type of a legal norm Chiticariu et al. (2013). Within the experiment, a domain expert specified patterns, which were coded into a formal pattern description language. The evaluation showed that there are certain classes, which could be extracted accurately, such as "Objection" or "Permission". However, several limitations were identified during the process and the implementation of the rules.

The evaluation already showed huge differences among the categories in terms of their classification accuracy. As mentioned above, classes such as "Objection" ,"Permission", or "Consequence" were identified with a reasonable F_1 score. In addition, there are classes with a lower F_1 score, but with a high precision, e.g., "Continuation", or recall, e.g., "Duty". Interestingly, while creating the rules there is a tradeoff between precision and recall in many cases. One can formulate the rules in such a way that it applies for a large set of norms, which increases the recall, but decreases the precision and vice versa.

The main problem with rule-based information extraction is the large linguistic variety of natural language. The attempt to formalize the expressiveness in patterns is very challenging and requires a careful and methodological approach. Patterns are formalized and once applied they either match or do not match a given textual phrase. However, if the text just slightly differs from the specified pattern, there will be no match. Those near misses cause problems because the existence of a single word, i.e., token, that does not change the semantics of a phrase can cause such a near miss.

Another challenge for the rules is the so-called domain portability. Usually, those rules are created for an information extraction task within a particular domain, e.g., tenancy law. In general those rules cannot be applied to another domain without major adaptations. However, within this research no attempt has been made to test the domain portability of the rules as formalized for the tenancy law.

Although the definition of the rules is a labor-intensive and tedious task, the efficiency with which those rules could be applied to the legal text was good. The classification was performed almost in real time without considerable latency or delays.

6.4. Performance Evaluation: Classifying Legal Norms with Active Machine Learning

Motivation

The analysis of statutory texts with regard to the semantic types of legal norms, is a desirable objective for the domain of legal informatics. Making the knowledge of domain experts explicit is required in order to enable computational legal reasoning and the extraction of legally relevant information from documents. In addition to the existing approaches on the classification of semantic types of norms using rule-based approaches, applying machine learning allows to extend the usage of technology even further. Although rule-based information extraction is still a commonly used technique for industry and sciences, the advances in machine learning are supposed to contribute to the negative aspects of rule-based information extraction and may help to diminish them.

Classification of legal norms can be addressed from different perspectives, e.g., from the philosophical, the legal-theoretical or, a the constructive one. With the stated motivation, a classification regarding functional aspects has been chosen. Functional classes of norms address the role of a norm within a given legal statute. These classes were derived bottom-up and harmonized within a taxonomy.

Objective

The objective of this evaluation is to assess the performance of active machine learning to classify legal norms from German statutory text regarding their semantic type, i.e., functional class. Thereby, the publicly available data was used, namely German federal laws from the domain of civil law.

The active machine learning approach was chosen in order to extend the rule-based information extraction and to contrast both approaches for information extraction. In addition, the active machine learning approach was compared against supervised machine learning without query strategies (random selection). This showed that using a query strategy is — at least for certain classifiers — superior to random selection strategies.

Data

The functional classification system divides legal norms into four types of statements: normative, auxiliary, legal-technical, legal-mechanism. The taxonomy differentiates normative statements into the following categories: statutory duty, statutory right, shall-do rule, and two types of consequence rules, namely positive and negative ones. Table 6.7 shows an overview of all classes.

The category of statutory duties further divides the subclasses order and prohibition. The class of statutory rights is decomposed into subclasses of permission and release. Auxiliary statements are divided into statements about terms and statements about norms. The first category can be divided further into explanatory, extending and limiting statements, in which the explanatory

Normative statement	Statutory duty		Order
			Prohibition
	Statutory right		Permission
			Release
	Shall-to-do rule		Shall-to-do rule
	Legal consequence		Legal consequence positive
			Legal consequence negative
Auxiliary statement	Statement about terms	Explanatory statement	Definition
			Specification
		Extension and limitation	
	Statement about norm	Legal validity	Legal validity and non-validity
		Scope of application	Temporal
			Personal
			Factual
		Area of application	Extension
			Limitation
			Definition
		Modification	
Legal-technical statement			Reference
			Continuation
Legal-mechanism statement			Procedure
			Objection

Table 6.7.: Functional type classification of statutory legal norms for German legislative texts (Waltl et al., 2017b). The table is organized as hierarchy of types being more general on the left and more specific on the right.

statements subsume the subcategories of definition and specification statements. The latter category is subdivided into modifications, legal validity, scope, and area of application categories. Norms, which are mainly legal-technical or legal-mechanism, were further differentiated into the categories of references and continuation, and into the categories of procedure and objection in the second case. The taxonomy was developed for German statutory legal norms focusing on civil law by two legal experts. The 22 different functional types and their hierarchical structure are shown in Table 6.7. The support of the different types varies heavily throughout the classes. The focus was to provide a comprehensive functional classification.

In order to prepare a suitable dataset for the norm classification experiment, one legal expert assigned a semantic type to every sentence of the tenancy law section in the German civil code (§535 - §595), published on March 1st, 2017. As every sentence was assigned to exactly one class, potential multi-labels have not been assigned. A norm was assigned to the class to which it primarily belongs. For example, if a statement is a right and a legal definition, the legal expert decided which was the primary role of the norm regarding its functional type and usage within the interpretation.

The overall task resulted in a gold standard consisting of 532 labeled sentences with 16 different

labels. As 14 of the 22 labels had less than 1,2% support, they were removed from the dataset used. The 504 remaining sentences were each assigned to one of the eight remaining classes. The classes and their distribution and within the gold standard is illustrated in Table 6.8.

Type (German)	**Type (English)**	**Occurrences**	**Support**
Recht	statutory right	126	25,00%
Pflicht	statutory duty	109	21,63%
Einwendung	objection	92	18,25%
Rechtsfolge	legal consequence	50	9,92%
Verfahren	procedure	49	9,72%
Verweisung	reference	46	9,13%
Fortführungsnorm	continuation	19	3,77%
Definition	definition	13	2,58%
		Σ 504	100,00%

Table 6.8.: Semantic types and their distribution within the manually labeled dataset.

This dataset was split into a test and a training data set. The test set consisted of 126 sentences (25%), which were randomly but representatively selected. The remaining 378 sentences (75%) were used as training set and iteratively provided to the active machine learning classifiers. The features provided to the classifiers were the tokens, in a bags-of-word representation and the corresponding POS information for each token.

Experimental Setup

The instances for the first learning round, i.e., the seed set, were randomly queried from the unlabeled training set. They have been labeled automatically according to the gold standard and used for training the classifier in the first round. For every subsequent learning round, the five most informative instances according to the query strategy were used. After each round, the trained model was applied to the test data to evaluate the performance of the current model. This process was repeated until all instances of the training set were labeled (72 learning rounds in total).

To compare the performance with the random selection of instances, the classifiers have been trained subsequently over 72 rounds. In each round, a randomly selected set of labeled instances was added to the training set. The training was implemented in such a way that the sequence in which the training data was provided was preserved.

The classifier NB was used with standard parametrization of MLLib. Due to performance reasons, the number of iterations for LR was decreased from 100 (default) to 10. The MLP had four layers, while the number of nodes on the two intermediate layers was 20 and 10, respectively. The size of the input layer was 2^{13} and the size of the output layer eight (i.e., number of types). The size of the seed set was 18 instances for each iteration.

Evaluation

The objective of this experiment was twofold, focusing on assessing

1. the overall quality of legal norm classification in German legislative texts using machine learning and to evaluate
2. machine learning performance using query strategies compared to the absence (random) selection strategies.

This was achieved by evaluating the model's performance by testing it after each learning round. To compare the performance of the ~acAML approach, standard evaluation metrics were used[10]:

1. Precision
2. Recall
3. F_1
4. Accuracy

Additionally, the performance was logged after each round to monitor and visualize the learning progress.

Each classifier was tested with each of the four implemented query strategies (see Section 3.5.5.1). No query strategy has been significantly superior compared to the others. Thus, the average accuracy was calculated combining the result of all query strategies. This was done for the classifiers Naive Bayes (NB), logistic regression (LR), and a multilayer perceptron.

The first result of the evaluation is shown in Figure 6.2. There, the x-axis indicates the learning progress as the number of instances labeled (relatively to the overall amount of training instances available), and the y-axis depicts the average accuracy of the classifier. It shows the performance of classifiers applying query strategies opposed to random selection machine learning.

Figure 6.2 shows that using query strategies is clearly superior to random instance selection when using NB and LR. The usage of query strategies boosted the learning behavior, so that classifiers are trained faster or with less instances required. In addition, they also resulted in a higher overall accuracy obtained during the classification process.

For the classifiers, the average accuracy was up to 10% higher compared to the random approach after a short "discovery phase". This can be observed for the NB and LR classifiers in the region between 20%-70% labeled instances. Additionally, the accuracy obtained was higher for all instances and, beside a few exceptions, never below the random selection machine learning. Within an increasing number of active machine learning rounds, the chance of overfitting increases as well. As one would expect the difference between the approaches diminishes after a large number of labeled instances (approx. 70%-95%). Finally, all the classifiers result in the same accuracy for both approaches: with and without query strategies.

The results in Figure 6.2 furthermore show the importance of having a "high quality seed". As

[10]Note: no binary classification

the seed set in this study was created randomly throughout different experiments, the learning differs significantly within the initial phases. After a discovery of the version space (so-called discovery phase), AML was significantly superior to classifiers using a random selection strategy. An improved coverage of the version space resulted in a higher accuracy while only a small set of the instances was labeled. In addition, the maximum accuracy of about 75% was achieved having labeled only 40% of the instances. This an increase of more than 6% requiring only 35% of the training instances.

A detailed inspection of the results is shown in the Figure 6.3. In this overview, the F_1 measure is depicted and differentiated by the individual classes. The detailed graphs are only shown for the best performing classifier, namely LR. For the analysis of the performance of individual classes, *consolidated evaluation measures* (averaging the results of all four query strategies) obtained by the LR are used. Figures 6.4 and 6.5 show the consolidated curves for the evaluation metrics precision and recall.

Several conclusions can be drawn from this qualitative inspection. While norm sentences be-

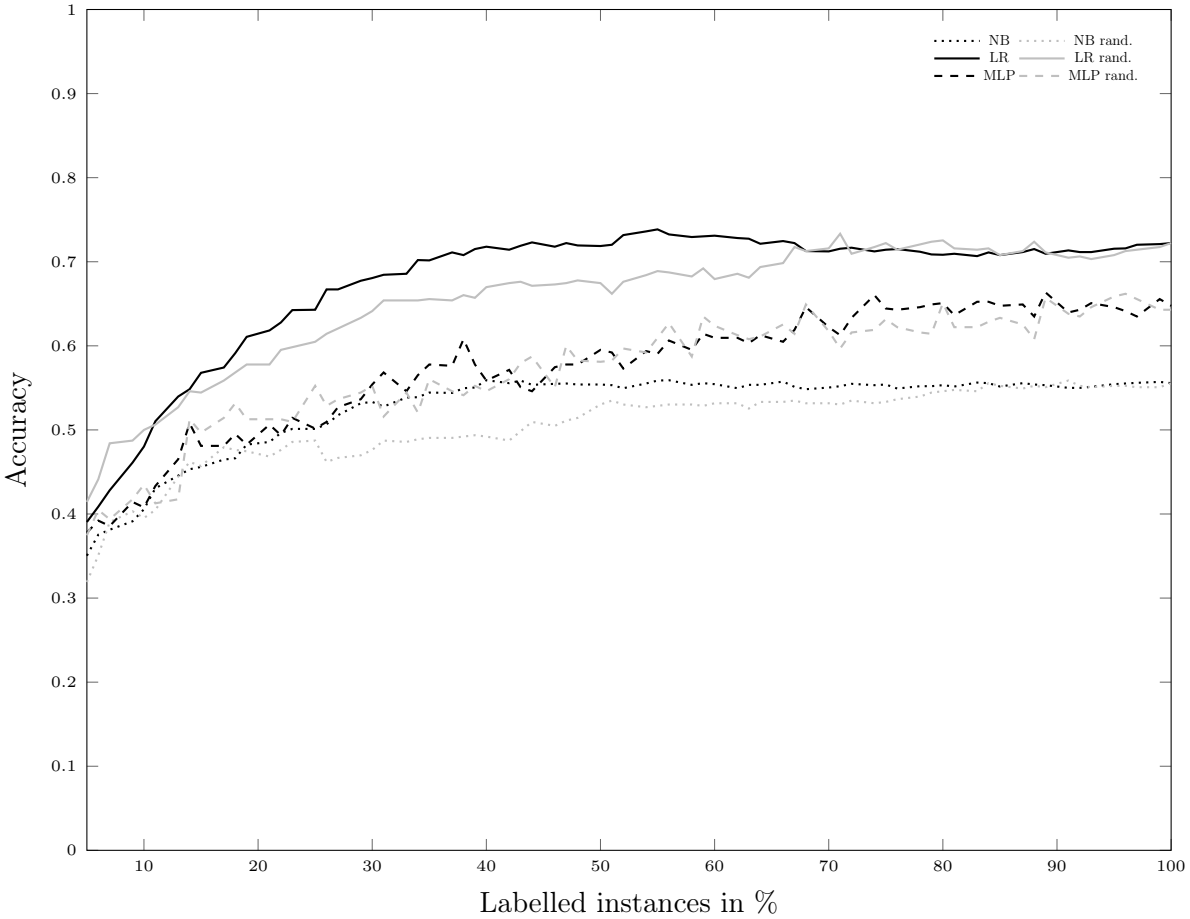


Figure 6.2.: Average accuracy of classifiers against random learning. Comparison of Naive Bayes (NB), logistic regression (LR), and multilayer perceptron classifiers (MLP).

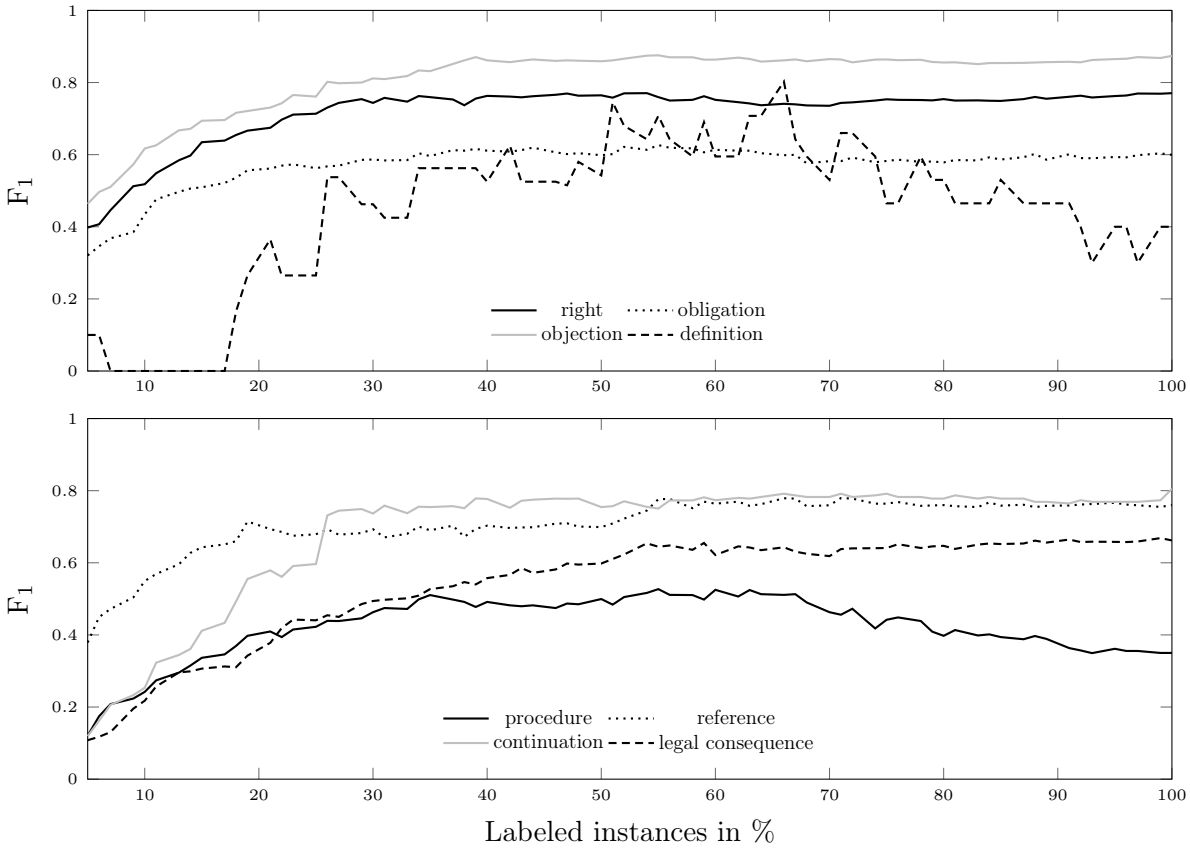


Figure 6.3.: Average F_1 per type using logistic regression classification.

longing to the semantic type *objection* are recognized very well, having an F_1 of almost 0.90 after 0.40 of training, norms referring to the type *definition* or *procedure* can hardly be classified. The reason for the low end-value for the type *definition* might be the lack of sufficient support within the selected training set (less than 3%). This results in a very small training set. In contrary, the training set for the type *continuation* contains only two more instances, and this type has an F_1 value of more than 0.80. This indicates that the classifier the lack of precision and recall is also due to other facts, such as a large linguistic variety or lack of proper features for this semantic type. Interestingly, the precision for the detection of *definitions* is very high, the recall is never above 50% and drops down to 22% towards the end of the learning. Seemingly, interferences with other types diminish the detection rate of definitions. A more detailed analysis on reasons for this phenomenon has not been performed within this evaluation.

However, considering the intermediate results, the types *continuation* and *legal consequence* have a very high precision and a good recall. The reason for the worsening results is more likely caused by the overfitting or interference with other semantic types. Overfitting can also be observed by inspecting the learning behavior for the type *procedure*. Although the overall performance is never very good (F_1 approx. 0.50 maximum), the performance is best during 40% to 65% of labeled instances. After all learning rounds, type shows the worst results, having both a low precision and recall. Although the number of training instances is high for the type *obligations*,

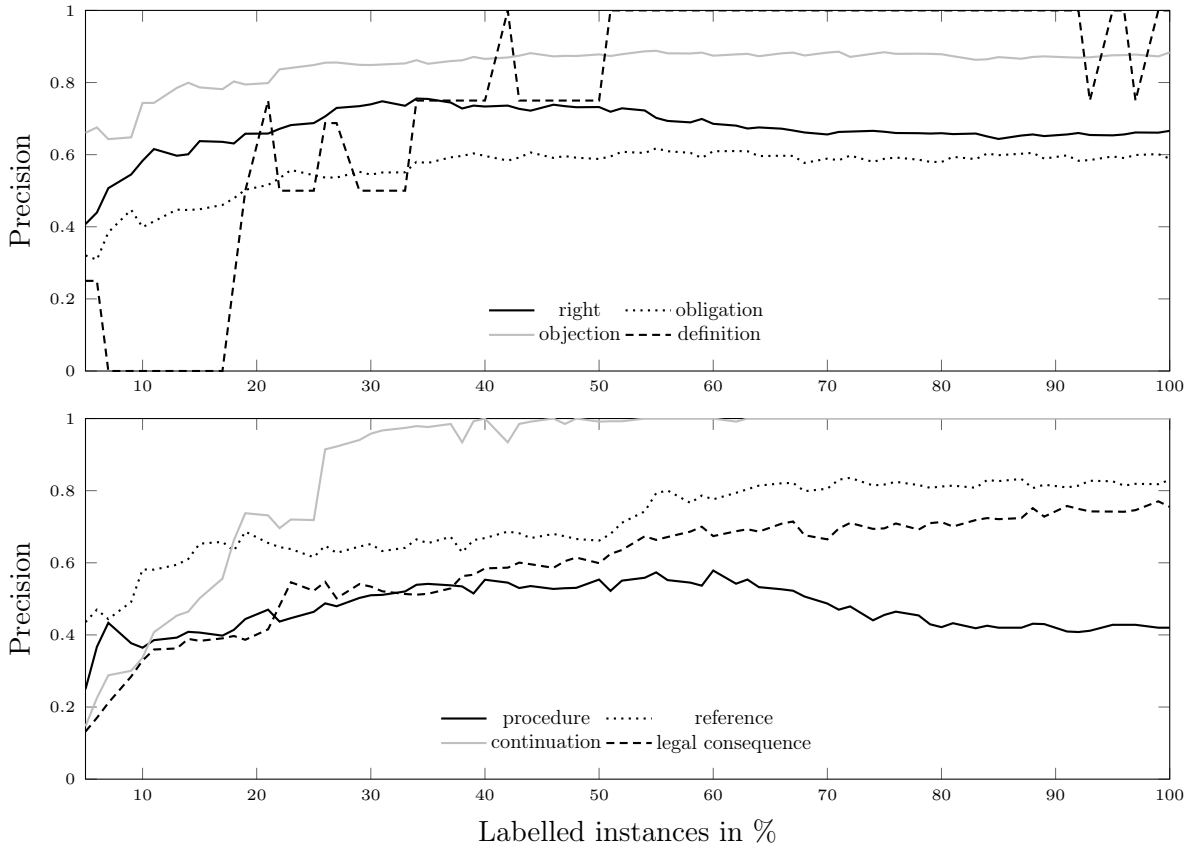


Figure 6.4.: Average precision per type using logistic regression classification.

the classifier only achieves an overall performance of F_1 approx. 0.60. The norm type *right* had the highest recall towards the end (95%), but a rather low precision (0.68).

Critical Reflection

This evaluation showed the performance of analyzing legal norms on the sentence level by classifying them into different categories with respect to their semantic type. Thereby, the semantic type follows the functional role of a norm with in a legislative text. The assessment revealed that AML using query strategies is superior to ML without query strategies. However, several open questions remain that have not been addressed with this evaluation.

The dataset was taken from a sub-domain of Germans civil law, namely tenancy law. It only consisted of 504 sentences, which have been labeled by one legal expert. The phenomenon of inter-annotator agreement has not been studied. With one domain expert labeling the data, the likelihood of consistency throughout the classes is increased. However, the dataset remains very limited. Having a small data set as used in this evaluation excludes a couple of machine learning techniques that are very common today, including artificial neural networks.

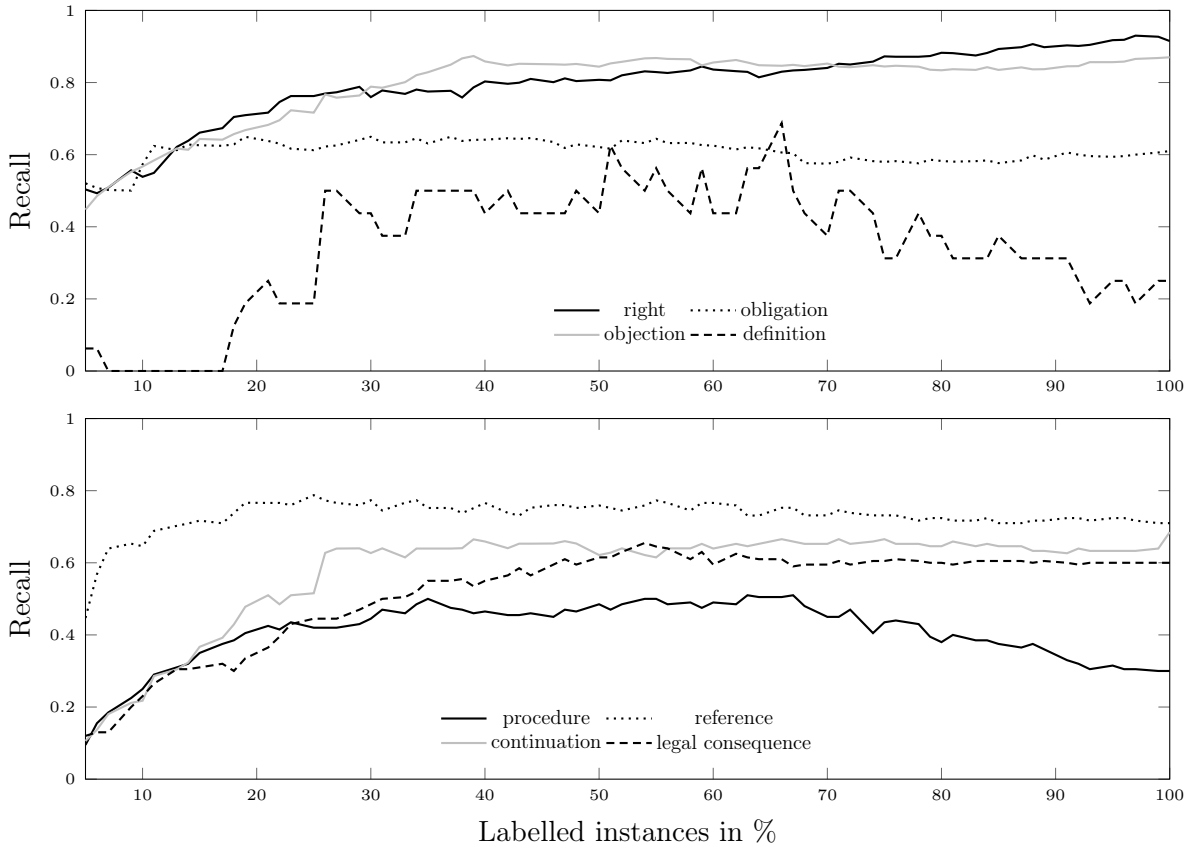


Figure 6.5.: Average recall per type using logistic regression classification.

The portability of the trained classifiers into another domain has not been studied either. It remains open whether the classifiers, trained for the tenancy law, works similarly well on predicting the class types of other, related legal domains, e.g., contract law or family law.

Regarding the analysis of the performance in the tenancy law, a more detailed error-analysis is required to improve the classifiers. The system has the possibility to provide more features for each norm, such as the context in which a sentence occurs, or additional linguistic features. Due to the complexity, these investigations have been left open for future research. This includes an analysis of decisive features, but also of parameters of classifiers and query strategies.

Beside to the open questions and potential directions for deeper analysis, the overall approach is a good completion to the rule-based approach of classifying legal sentences into different categories and types. Combining them, as conceptually proposed by Waltl et al. (2017b) seems to be a way forward in semantically analyzing legal norms.

6.5. Formalizing Termination Notice Periods of Germany's Tenancy Law

Background and Motivation

The German tenancy law is part of the civil law and codified in the German Civil Code §§ 535 – 597. The law regulates the legal framework based on which rental of goods, i.e., lease, can be agreed on. These are basically bilateral agreements between a lessor and a lessee. Just as both parties agree on the applying terms and conditions both parties have the right to terminate the lease. To settle this the German legislator has specified a couple of legal norms that state the conditions that need to be considered. This case study formalized a particular set of conditions, namely the termination notice periods of leases for an indefinite period of time. These are highly relevant for German citizens, as the agreement on renting residential space for an indefinite period of time is very common.

Within the German Civil Code, the appropriate norms are in the second book, title 5 "Lease, usufructuary lease", chapter 5 "termination of the lease", subchapter 2 "leases for an indefinite period of time": §573 and §573c. An excerpt of the sections is stated below.

German Civil Code §573: Notice of termination by the lessor

(1) [...] termination for the purpose of increasing the rent is excluded.

(2) A justified interest of the lessor in the termination of the lease exists, without limitation, in cases where

1. the lessee has culpably and non-trivially violated his contractual duties,

2. the lessor needs the premises as a dwelling for himself, members of his family or members of his household, or

3. the lessor, by continuing the lease, would be prevented from making appropriate commercial use of the plot of land and would as a result suffer substantial disadvantages; the possibility of attaining a higher rent by leasing the residential space to others is disregarded; [...]

German Civil Code §573c: Termination notice periods

(1) Notice of termination is allowed at the latest on the third working day of a calendar month to the end of the second month thereafter. The notice period for the lessor is extended, by three months in each case, five and eight years after the lessee is permitted to use the residential space.

The essence of the two norms, namely to decide whether a termination is valid or not, can be formalized into a model-based decision structure to foster its operationalization and execution.

Objective

The objective of this case study is to show the feasibility of model-based reasoning to formalize complex decision structures of German legislative texts. The implemented model should be applicable and executable to support during decision-making processes on deciding whether a termination is valid or not.

Models are always intended to serve a particular purpose. Based on this case study, different conceptual decisions, such as the granularity of modeling, and the modularity of types, and strategies to deal with uncertainties and vagueness within legal texts are discussed.

Formalization

The two sections of the German civil code state different conditions that need to be fulfilled for a termination to be valid. These conditions can either be evaluated as true or false, i.e., Boolean values, or can be represented as numerical expression on date values. In a manual process, the semantics is captured into a model based on the representation consisting of types, attributes, and relations.

Semantic Model

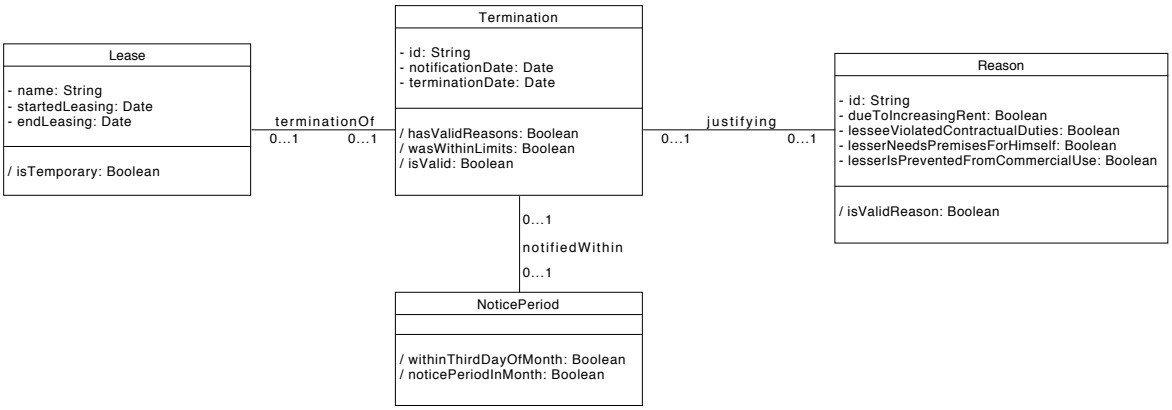


Figure 6.6.: Facts focusing on termination periods and justification.

The semantic model of the situation for the termination of a lease by the lessor is shown in Figure 6.6. Therein, four different types were identified:

1. **Termination:** The actual termination object, the validity of which should be confirmed or rejected.
2. **Lease:** Describing the object that is leased including the information of agreed start and end date.

3. **Reason:** The termination has to be justified. Therefore, several reasons are legally valid, while others are not. Among other issues, this serves to avoid arbitrariness of the lessor and protects the lessee.
4. **NoticePeriod:** The time constraints that exist for the termination object. These depend on the time a lessee has been leasing the lease object.

The relations among these objects are chosen in a way that the Termination type is central. Lease, NoticePeriod, and Reason relate to these as they contribute to the overall validity of the termination. The multiplicities are 0...1 in every direction, as no type must necessarily refer to another type, but would make the termination invalid by default. For example, the absence of reasons would automatically invalidate the termination.

The relations can be formalized as follows:

$$\begin{aligned}
&terminationOf \subseteq Lease \times Termination : \\
&(l,t) \in terminationOf \implies Lease\ t\ should\ be\ terminated\ by\ Termination\ t \\
\\
&notifiedWithin \subseteq NoticePeriod \times Termination : \\
&(n,t) \in notifiedWithin \implies NoticePeriod\ n\ applies\ to\ Termination\ t \\
\\
&justifying \subseteq Reason \times Termination : \\
&(r,t) \in justifying \implies Reason\ r\ justifies\ Termination\ t
\end{aligned} \tag{6.1}$$

Executable Model

In addition to the semantic model elements, the derived attributes formalize the executable semantics of the model. They take into account the existing types, attributes, and relations; and they reason over the instances and their attributes. Thereby, seven different derived attributes were formalized:

$$\begin{aligned}
t : Termination &\implies t \text{ is instance of } Termination \\
l : Lease &\implies l \text{ is instance of } Lease \\
n : NoticePeriod &\implies n \text{ is instance of } NoticePeriod \\
r : Reason &\implies r \text{ is instance of } Reason
\end{aligned} \tag{6.2}$$

The Equation 6.2 instantiates and assigns a concrete type to the variables t, l, n, r.

$$l.isTemporary = \begin{cases} true, & if\ l.endLeasing\ is\ null \\ false, & otherwise \end{cases} \tag{6.3}$$

$$t.hasValidReasons = r.isValidReason \quad \text{for } (r,t) \in justifying$$

$$t.wasWithinLimits = \begin{cases} true, & if\ t.terminationDate - r.noticePeriodInMonth \geq 30 \\ false, & otherwise \end{cases} \quad \text{for } (n,r) \in notifiedWithin$$

$$t.isValid = r.hasValidReasons \wedge r.wasWithinLimits \tag{6.4}$$

$$\begin{aligned} r.isValidReason = \neg r.dueToIncreasingRent \\ \vee\ r.lesseeViolatedContractualDuties \\ \vee\ r.lesserNeedsPremisesForHimself \\ \vee\ r.lesserIsPreventedFromCommercialUse \end{aligned} \tag{6.5}$$

$$n.withInThirdDayOfMonth = t.notificationDate.day \leq 3 \quad \text{for } (n,t) \in notifiedWithin$$

$$n.noticePeriodInMonth = \begin{cases} 8, & if\ t.notificationDate.Year - l.startedLeasing.Year \geq 8 \\ 5, & if\ t.notificationDate.Year - l.startedLeasing.Year \geq 5 \\ 2, & otherwise \end{cases} \quad \text{for } (n,t) \in notifiedWithin\ and\ (l,t) \in terminationOf \tag{6.6}$$

The Equations 6.3 – 6.6 formalize the executable semantics between the attributes and derived attributes. The formulas have been expressed in MxL to enable automated reasoning. In addition, the formalization allows to access the AST for the inspection and analysis of the decision structure.

The visualization of two explanations is shown in Figure 6.7. Thereby, the type information, the reasoning expression, and the AST are depicted. The logical expressions for both derived attributes follow a classical decision trees, in which the leaves are attributes and the nodes form the logical connection between them.

Execution & Instantiation

Once the model is created, it can be executed, i.e., instantiated, via the knowledge acquisition component. Therefore, the facts and evidence are provided to the reasoning component, based on which the inference engine draws the conclusions for the concrete case.

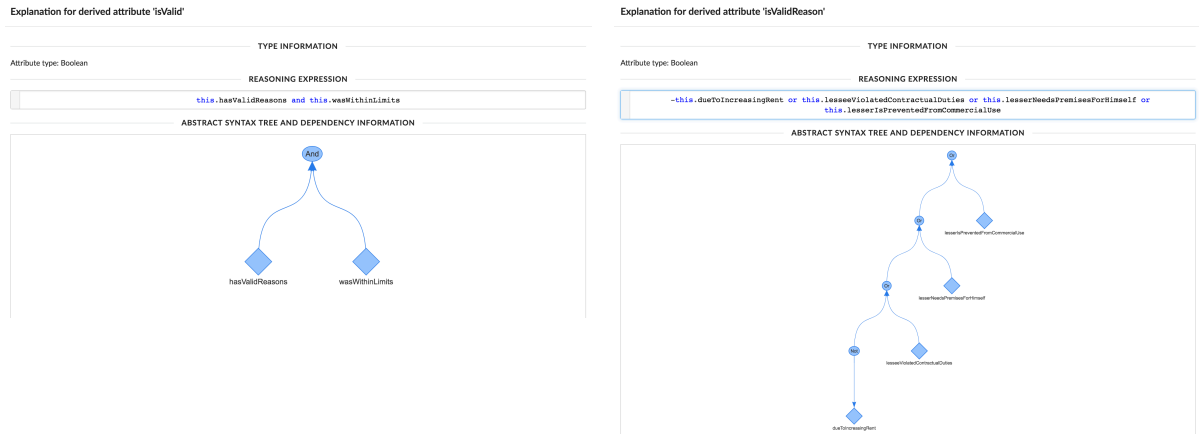


Figure 6.7.: Explanation for two derived attributes: isValid and isValidReason.

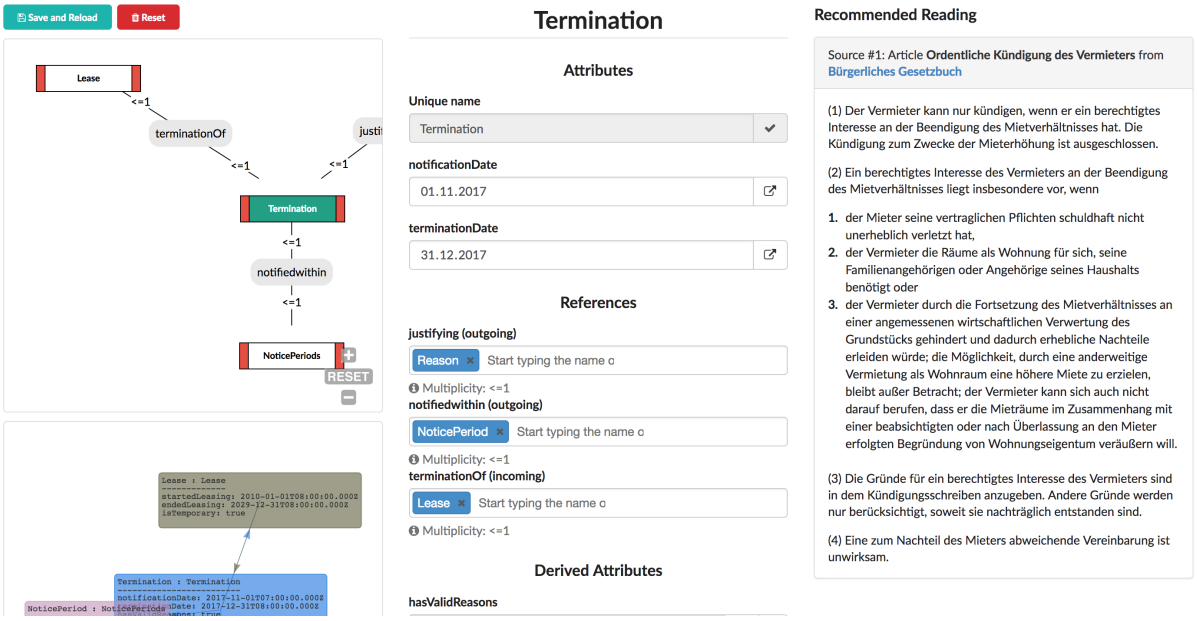


Figure 6.8.: Knowledge acquisition interface for the termination type.

Figure 6.8 shows the knowledge acquisition component for the termination type. The left part of the system visualizes the model, the middle part creates the input fields to insert the facts, and the right part provides text fields with recommended readings, which are basically the parts of the legal document that was underlying the interpretation process.

The values in the figure refer to the termination type, with the name “Termination”. A user has already inserted the notificationDate (01.11.2017) and the terminationDate (31.12.2017). Additionally, the references to the lease type, the reason type, and the noticePeriod type have been set. For the remaining types, which are not shown in the figure, the attributes have been set as well.

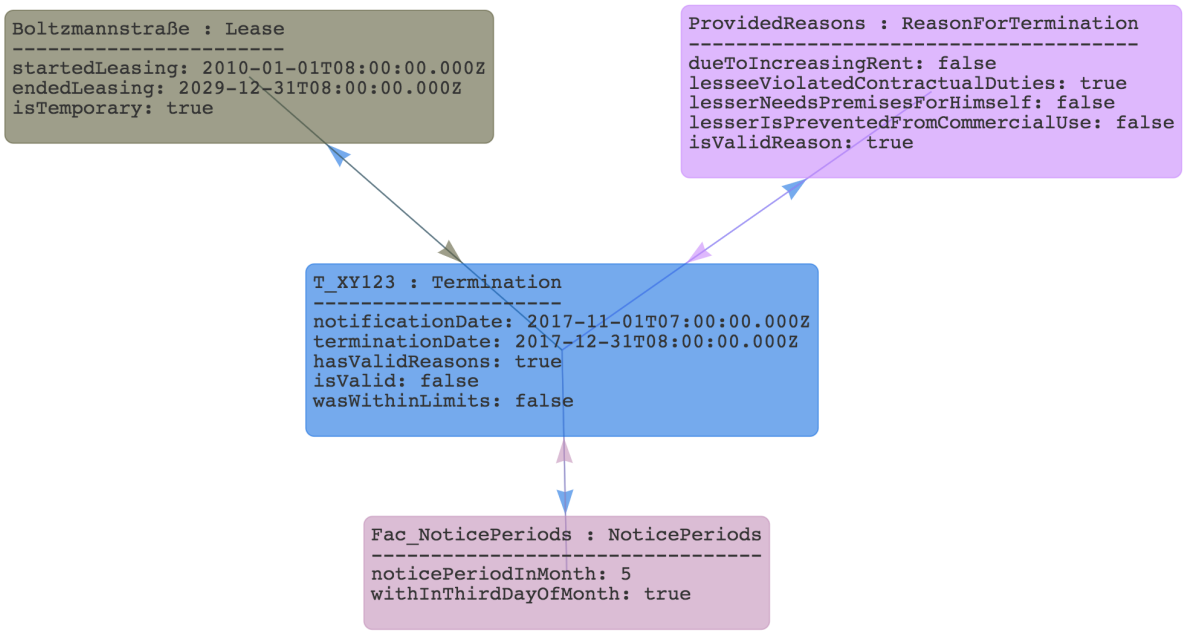


Figure 6.9.: Instantiation of the semantic model showing the atomic and derived attributes, as well as relations among instances.

The full instantiation can be seen in Figure 6.9. The instantiation shows the four different instantiated types:

$$\begin{aligned} \underline{Instance} &: \underline{Type} \\ T\_XY123 &: Termination \\ Boltzmannstrasse &: Lease \\ Fac\_NoticePeriods &: NoticePeriods \\ ProvidedReason &: Reason \end{aligned} \tag{6.7}$$

The values of the attributes, relations, and derived attributes are shown in the figure. The inference engine determines the value for every derived attribute. It can be seen that the value for the derived attribute “isValid” (see Equation 6.4) is “false”. Although the termination has valid reasons (lesseeViolatedContractualDuties is true), the notice period as required by law has not been considered adequately. The notification date for termination was within the first three day of a month, however, the termination date lies only 3 months ahead, which is not allowed. The lease object has been leased for seven years (since 01/01/2010, according to the facts), and by law, after 5 years of leasing, the termination period has to be at least five months ahead. Hence, the instantiation view, which is automatically created, gives an instant overview of all the provided facts and their evaluation. It allows to validate and analyze a given case.

Based on this formalization of a small but highly relevant section of the German tenancy law, the potential of model-based reasoning has been shown. It highlights the benefits of reasoning on

logical and arithmetical expressions and combines these with structuring the domain knowledge by applying techniques for knowledge representation, such as ontologies. However, a couple of drawbacks and limitations exist within this approach, which will be investigated in the next section.

Critical Reflection

The critical reflection for the modeling part is subdivided into three different areas, namely logic, modeling, and functionality.

Logic

From a formal point of view, the MxL fulfills certain desirable properties, such as propositional and arithmetical reasoning. In addition, it is a higher-order logic and can easily be extended with custom Java functions.

However, it does not natively support temporal reasoning. Explicit expressions of time and calculations, as well as comparisons are valid, but expressing propositions that need a qualified temporal layer are not possible. These include classical temporal operations like "until" or "release".

In addition, defeasible reasoning, which is particularly relevant for argumentation logics, is also not supported. The approach does not allow the specification of contradicting or conflicting rules, unless their conflicts are explicitly resolved, for example in case differentiation.

Until now, the system is focusing on formalizing statutes and has not been extended for case-based-reasoning. Support of case-based-reasoning would be highly relevant to provide decision support for indeterminate or vague (legal) terms, which are widespread in German laws. Currently, vague attributes are modeled as explicit and their final value, e.g., true or false, needs to be decided by the user providing the data within the knowledge acquisition phase.

Modeling

The fact that the basic structure always has to comply with the model-based approach, which requires a matching with types, attributes, relations, and derived attributes, is a constraint that must not be ignored. Pure logic programming, for example, allows for unconstrained decision structures. The modeling approach as such does not contribute to the expressiveness of the logical formalism as such, although it reflects real-world entities.

As models are always designed to fulfill a pre-defined purpose and as there are different design decision during the modeling process, a text can be interpreted and formalized into different models. Semantically equal models can look differently. For example, the formalization for the notice of termination by the lessor as shown in Figure 6.6 uses a separate type for the notice period. This would not be necessary, as all its derived attributes could also be expressed in the termination type. The definition and analysis of best practices and patterns for semantic

modeling has not been studied in the thesis at hand. However, interesting follow-up questions arise instantly, which can be addressed by further research.

Separating the modeling process from its execution requires a sound and consistent model. Contradictions in the model cannot occur, as expression language would not allow to express them. Hence, consistency by design is enforced, which is an additional restriction for the overall modeling approach.

Functionality

In terms of functionality it is difficult, and most likely impossible, to ex ante define all the different functions that are required to fully formalize the German law. This includes functions to determine certain properties or values, such as checking whether a given day is a weekday or is within the first three working days of a month.

The determination of those functions and their subsequent implementations are left open for further research. The systems modularity fosters the integration of new functions and offers the expressiveness of the Java programming language for new functions and operators.

CHAPTER 7

Conclusion

This chapter reflects on the contribution of this thesis and summarizes its results. Additionally, successive and emerging research questions for further research are briefly outlined and discussed.

The chapter is structured in three parts: Section 7.1 summarizes the main outcome of the thesis and discusses the results with respect to the initially formulated research questions. The critical reflection of the results is provided in Section 7.2. The limitations are analyzed in five subsections, each of which focuses on a different aspect of the overall concept and implementation. Finally, Section 7.3 describes different follow-up research paths to be followed based on this thesis. These cover technical approaches, but also challenges regarding legal science and practice.

7.1. Summary

The overall objective of the thesis is summarized within the main research hypothesis, as stated in Chapter 1:

> **Research hypothesis:** A collaborative software environment can support the semantic annotation of legal documents using text analytics components to design and formalize computational models within ontological decision structures.

To verify (or falsify) this hypothesis was narrowed down into sub-problems, which could be handled efficiently. This led to the structure reflected within the thesis: seven chapters, describing different aspects and contributions of the overall research.

In Chapter 1, this thesis establishes the scientific problem of legal text analytics and of compu-

tational models for legal reasoning. Based on the reasonable assumption that the work of legal experts, i.e., scientists and practitioners, is knowledge-, time-, and data-intensive, the chapter established the use of technology as a means to support them in these processes. According to commonly accepted literature studies and recent insights, two main research directions address the analysis of legal documents using software and the representation of interpreted legal decision structures for computational reasoning. Within this section the research hypothesis was divided into eight different but connected research questions.

In order to understand the state-of-the art in software support for the legal domain, a summary of previous research was provided in Chapter 2. The chapter described different approaches that have successfully been applied to semantically analyze legal documents. The focus within this section was not only to investigate different use cases, but also to describe the concrete software technical implementations that have been applied. Based on this consideration, a main research gap was derived, namely the lack of a generic data analysis framework for legal documents. Although several different attempts have been made to extract and structure legal documents along different criteria using software tools, the community lacks a framework that can be extended to support different use cases, at the same time allowing the reuse of already implemented components. This would decrease the effort of reimplementing frameworks and lower the barrier for extensive research. The second main focus of Section 2 was to provide an overview of approaches in model-based, i.e., ontological, legal reasoning to formalize of legal decision structures. Since the adoption of the semantic web as a standard in structuring knowledge in the web, the idea to create a formal representation for legal knowledge is highly attractive. The main contributions were discussed and regarding their relevancy for this research, explained in detail.

Based on the prior scientific contributions and the identified research gap, the framework for collaborative data science, i.e., text analytics, for legal documents was conceptualized in Chapter 3. The main focus was to describe an architecture to implement three main software technology principles: i) modularity of components, ii) expandability regarding additional analysis functionalities, and iii) the providing collaborative features. The collaborative aspects were covered via the implementation of the framework within a Java web application. The architecture needed to support the management of the legal documents, i.e., mainly composed of textual information, and the processing of legal documents, i.e., application of software components for semantic analysis, including the extraction of structured information from the text. For the management of legal documents, a Java implementation with a generic data model was designed. The extraction processes were performed within a modular Pipes & Filters architecture. Based on the extensive analysis and comparison of different frameworks, the Apache UIMA was identified to be the most suitable framework for this task. It allows an easy reuse of components and the configuration of robust pipelines to analyze legal documents in a web application. Furthermore, the access for machine learning functionality was provided by an additional service, which was built on Apache Spark. The focal point thereby was the creation of a system, that would learn based on the interaction, i.e., supervised machine learning, with the system during manual interpretation and analysis of legal documents.

In Section 4, the main idea of combining knowledge engineering and computational reasoning was developed and extended beyond the current state-of-the-art. Whereas previous approaches focus on the ontological modeling and the usage of description logic to formalize dependen-

cies among the entities and types, the proposed method extended this by adapting an existing domain-specific language that allows for higher-order logical and arithmetical reasoning (see Section 4.2). In doing so, a known limitation of description logics, such as OWL, was overcome. The insights from knowledge engineering were incorporated by providing means to formalize interpreted legal norms into computational models consisting of types, attributes, and relations. During the interpretation phase, the user could formulate executable representations supporting computational reasoning. The executable semantics was formalized as a Model-based Expression Language expression, which has extensively been studied in a research group since 2013 (see Reschenhofer 2013). The model-based reasoning approach was conceptualized for a full-stack implementation, including a model-store, inference engine, and interaction components. An extensive discussion of components to increase the transparency of automated decision making was provided (see Section 4.4), resulting in the implementation of three different components to analyze the formalized and instantiated decision structure: instance and fact view (see Section 4.4.1), abstract syntax tree (see Section 4.4.2), and data flow (see Section 4.4.3).

Based on the conceptualization as provided by the Chapters 3 and 4, the concrete implementation was illustrated. It described the main parts of the implementation structured along the three main contributions: the framework for processing of legal documents using a text analysis engine (see Section 5.2); the active machine learning component, which is part of the text analysis engine (see Section 5.3); and the model-based reasoning component, which is mainly consisted of the provision of interaction components and the integration of an existing meta-model based information system (see Section 5.4). The main challenge was to follow the design principle of modularity to foster the implementation of new functionalities without unintended changes in or side-effects of the existing methods. The technology used to integrate different software components within a processing pipeline served as an example to display the modularity. Along this pipeline, different semantic operations, such as tokenization, POS tagging, or pattern detection, were performed. Each software component added information, which was used by subsequent components. This modularity ultimately led to a generic software framework, allowing components to be exchanged flexibly. This could be used to extend the analysis either to new document types, or to documents written in another language. In addition, this decreased the barrier of exchanging software components among research groups.

The evaluation Chapter 6 was structured into four different and complementing sections. The main idea was to assess the quality and applicability of the prototype in the individual aspects. First, Section 6.2 described a case study, that was performed with an industry partner on the software-supported analysis of tax law judgments to assist editorial processes on automatically extracting information from legal documents. The results showed that for narrow and well-defined use cases, a high precision (0.94) and recall (0.90) could be achieved ($F_1 = 0.92$). Then, Section 6.3 describes the rule-based information extraction performance for classifying legal norms into different functional categories. The usage of rules to extract information from textual documents is still predominant in practice (see Chiticariu et al. (2013)); however, their problems and limitations are hardly discussed in current academic research. Following these considerations, legal norms, based on their functional role within the statutory text, were also classified using a supervised (active) machine learning approach (see Section 6.4). We showed, that active machine learning was superior to classical supervised machine learning in text classification. For several classes, i.e., rights and objections, a high accuracy (> 0.90) was achieved.

Classifiers showed problems with classes that rarely occurred among the training data set, e.g., legal definitions. Finally, the formalization of the legal decision structure was performed for a small but relevant problem in the German tenancy law: the termination period for the cancellation of a rented apartment (in Section 6.5). The potential of structuring the decision structure into types, attributes, relations, and derived attributes was shown.

Based on the main objective of the thesis, the research hypothesis was divided into several individual research questions, thus allowing a structured approach for the investigation. Those eight different research questions were formulated that are subsequently addressed and answered within this thesis. In the following, we recall these research questions, complemented by potential answers that could be deducted from the results generated in this thesis:

> **Research question 1:** What is the state-of-the-art in software-supported analysis of textual documents in the legal domain focusing on legislative and judicial texts?

Based on scientific literature, the state-of-the-art analysis was assessed considering most recent publications and projects in the field of legal informatics. The most common technological frameworks used to semantically analyze legal documents are Apache UIMA or GATE. However, in many research projects, no efforts are made at all to create an advanced legal text analytics framework, that might serve as an ecosystem or platform for the analysis of legal documents. It seems possible that research tries to avoid the "overhead" of creating such a platform, although the effort would pay-off in the long term. From the technologies used, the application of a broad variety of different methods from regular expression to neural networks can be observed (see Section 2.1 and Section 2.2).

> **Research question 2:** What are the methods and tools to formalize ontological decision structures emerging from statutory texts?

Many different attempts have been made to formalize statutory texts, which showed the challenges and the great potential thereof. Regarding the formalization of decision structures, deductive reasoning systems including non-monotonic and temporal reasoning, can be considered state-of-the art. Ontologies are commonly accepted to structure legal knowledge, but there seem to be no serious attempts to use description logic in order to perform advanced legal reasoning. This is certainly also due to the lack of functionality as provided by description logic, e.g., no temporal reasoning, no arithmetic reasoning. The latest trend focused on the provision of an interchange format to describe legal knowledge, such as LegalRuleML (see Section 2.3).

> **Research question 3:** What could a reference process, considering activities, roles, artifacts, and software tools, for the software-supported semantic analysis of legal documents look like?

In order to fully exploit the potential of software-supported semantic analysis the thesis proposed a reference process model that differentiates between activities, roles, artifacts, and software support & tools (see Section 3.1). The process, with its iterative activities, allows a structured approach by guiding through the different phases and by highlighting the importance of different software components within them. The process has been used and refined within a case study that was performed with an industry partner (see Section 6.2). The main activities are: import, analysis, and application. The analysis phase consisted of three iterative activities, namely

"Refinement & creation of the model", "Transformation and operationalization of the model", and "Evaluation of the operationalized model".

> **Research question 4:** What are the design principles and components of a software architecture enabling a collaborative environment for information extraction processes from legal documents?

Based on the reference process model, a modular software architecture for legal text analytics was developed (see Section 3.4). The architecture allows the usage within a collaborative environment, i.e., a web application. The architecture consists of six different main components: import, export, data storage, data access layer, user interface, and a text analysis engine. Each of the main components is composed of other components. The exchange of information is handled via well-defined interfaces to guarantee the modularity of the components. In order to ensure the modularity of software components within information extraction processes a Pipes & Filters architecture was used (see Section 3.5). Based on extensive comparison of software frameworks the Apache UIMA was adopted for the usage within the research prototype. Apache UIMA manages the configuration and application of processing pipelines, consisting of the subsequent application of software components, i.e., annotators. The framework is robust and thread-safe, which makes it particularly suited for the usage within a collaborative web application.

> **Research question 5:** What are the elements of a reference process with the aim of formalizing of statutory texts into computational decision structures?

Based on the semantic analysis of statutory texts, the interpretation, i.e., formalization, of statutory texts should be supported. In analogy to the reference process model for semantic analysis, a reference process model for the interpretation of statutory texts was derived. The reference process differentiates between activities, roles, and tool-support; and it is organized in four subsequent phases: import, analysis, interpretation, and application (see Section 4.1). The interpretation of a statutory text is formalized within a model-based, i.e., ontological, decision structure including types, entities, and relations among types.

> **Research question 6:** What are the components of an ontological reasoning framework modeling the computational semantics and connecting the interpreted legal texts with the corresponding model element?

Reasoning on model-based, i.e., ontological, decision structures requires a modeling environment that allows the creation of the ontological entities, namely types, attributes, relations and derived attributes (see Section 4.2.1). For this purpose, an existing meta-model based information system was used. The reasoning was done using a Domain-Specific Language (DSL), which already existed but was adapted for the usage for reasoning. Every ontological entity can be connected with the analyzed and interpreted legal texts, which allows provenance of particular parts of the reasoning structure and tracing it back to the textual source in the law from which it emerged. The connection is done using annotations, which are not only the result of the text analytics process, but are also the central element in linking text with the decision structure (see Section 4.3.2).

> **Research question 7:** How can the implementation of a model-based reasoning

framework capturing computational semantics of interpreted statutory texts look like?

The thesis describes a full-stack implementation of the model-based reasoning, consisting of a model and fact storage, a model execution component, and an interaction component (see Section 4.3). The model and fact storage persists the interpreted model in a representation that is suitable for computational reasoning. The model execution component can operate on the model and derive new knowledge. In addition, it allows for the analysis of dependencies and inferences, which enables the system to provide an explanation for the inferred knowledge (see Section 4.4). Finally, the interaction component allows to formalize an interpretation and to add new knowledge, based on which the system can make inferences.

Research question 8: What accuracy can be achieved during information extraction and norm classification in judicial and statutory texts? What are the limitations and emerging research directions?

The performance of rule-based information extraction and active machine learning to classify sentences of German statutes according to their semantic type, i.e., functional role, showed that the accuracy heavily varies throughout different semantic types (see Sections 6.3 and 6.4). High linguistic variety and a very unbalanced training data set can be considered the main challenges for the information extraction using (active) machine learning.
The applicability of the model-based reasoning approach was shown using proof-of-concept for the child benefit regulation from the German tax law, and for the notice periods regarding the termination of a rented object in the German tenancy law (see Section 6.5).

Based on the concrete answers for the research questions, that were formulated to answer the overall research hypothesis, a critical reflection is provided in the next section.

7.2. Critical Reflection

Understanding the functionality of an implementation implies to understand the limitations of a prototype, and of the research conducted and described within this thesis. This chapter is dedicated to describing the main limitations; for this purpose, it is structured into four different sections. In Section 7.2.1, the main limitations of the text analytics framework as implemented are discussed. Section 7.2.2 reflects on the main limitations for the model-based reasoning and formalization. In Section 7.2.3, the main concerns regarding the end-user applicability are discussed. Based on these considerations, the evaluation as performed within this thesis is critically discussed in Section 7.2.4. Finally, Section 7.2.5 reflects on the used research method, namely the usage of design science within the thesis.

7.2.1. Functional Limitations of the Legal Text Analytics Frameworks

Due to the implementation several technical limitations for the text analytics framework exist. These can either be considered as severe ones, for which the way the framework was implemented does not offer a solution, or minor ones, for which — most likely — a solution can be found.

One main limitation that arises from the design decision to use the Apache UIMA framework is that the UIMA is a document-centered analysis framework. The pipeline is configured and instantiated for each document and the different documents are processed individually. This offers the advantage that the system can scale very well and can be used in a multi-threaded environment; however, if there is a task to determine dependencies between documents, these cannot be established easily. It would require an additional resource, to be maintained outside of UIMA, which could store the this additional, non-volatile information about a document.

The analytics components lack linguistic operations that would be very helpful during the analysis of legal documents. Two unavailable components (or insufficiently accurate) components are the parsing of sentences (see Section 3.4.1.4) and a linguistic component to detect auxiliary sentences. The latter ones are very common in statutory texts, but intensify the issue with assigning a functional type to the sentence, as it may contain more than one semantic role. In addition, it is not easy to determine the main subject of a sentence if it contains auxiliary sentences.

As each pipeline is configured to subsequently apply annotators to achieve different annotations tasks, such as sentence splitting, POS-tagging, or complex pattern matching, the annotators cannot perform their tasks without errors. No annotator works at 100% precision and recall. This fact constantly diminishes the overall precision and recall of the tasks, e.g., extraction of meta-data as described in Section 6.2. Savelka and Ashley (2017) showed that even splitting of sentences in texts from the legal domain cannot be solved at a very high accuracy. The same holds for the components as implemented the prototype. However, no extensive evaluation was performed to provide reliable numbers for the accuracy for every sub-component.

The third limitation of the text analytics component concerns the performance in terms of computational complexity. As most of the annotators need some time for their annotation, at least for sufficiently large documents, and since the UIMA framework adds additional management overhead on top, the analysis can no longer be performed in real-time.

The last limitation of the prototype is its implementation in Java. Although this comes along with the desirable qualities of having a large community and a huge number of frameworks and libraries one can reuse, it is not easily possible to re-use latest implementations in the field of ML as these are mainly composed in Python[1].

7.2.2. Functional Limitations of the Model-based Formalization

The computational semantics is not only defined in statutory texts, i.e., laws, but requires the consideration of additional literature, e.g., commentaries. Unfortunately, only a few laws are written clear and unambiguous enough to be suitable for formalization. Surden (2012) describes different requirements for computational law, one of them being data-orientation and shared meaning of words. However, those requirements are hardly met in statutory texts. Instead, the intrinsic vagueness and open-texturedness of normative texts intensifies the problem of formalization. Formalization approaches in combination with the model-based approach unveil unclear and vague regulations, and can make those explicit to the user. However, this does not solve

[1] see for example the scikit learn framework http://scikit-learn.org/, accessed on September 3, 2018

this issue for him. Interpretation is still required and might not be possible without assumptions that go beyond the text as provided by the legislator.

The model-based expression language heavily depends on the subsequent modeling step. The underlying model predetermines the structure consisting of types, attributes, derived attributes, and relations. The system does not enable users to model intrinsic legal concepts, such as deontological concepts.

In addition, the MxL is a functional expression language without side-effects, and it imperatively defines exactly one derived attribute. Consequently, the inference engine follows a classically deductive monotonic logic approach. At the current state, no operators are implemented into the system supporting probabilistic reasoning (e.g., Bayesian Networks). How the system could be improved in order to support probabilistic reasoning, focusing on the aspect of user enablement, is a very interesting research question (see also Timmer et al. 2015).

The system has not been designed to reflect temporal logic. Instead, the system supports a lifecycle management for the creation and maintenance of the models. This reflects the lifecycle of textual documents that underlie the creation of models, i.e., interpretation. However, as MxL supports reasoning on dates (as native datatype) and also has a NOW function, it is possible to take a basic temporal dimension into account during the reasoning.

The system does not support abductive or other advanced reasoning principles that have been investigated throughout the last decades and that are potentially relevant for legal reasoning (see also Bench-Capon et al. 2012).

7.2.3. User Applicability

Throughout the thesis the user has always been taken into account. However, Ko et al. (2011) state that "End-user development has been defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact". For the contribution as provided by this thesis, this would mean that, on the one hand, end-users can train and apply linguistic models to process legal documents, thereby supporting them during reviewing and analysis. And on the other hand end-users should be enabled to capture the interpreted decision structures in computational models that can be executed.

In order to critically reflect on the end-user applicability for the legal text analytics framework one has to admit that although the web application provides a user interface that enables end-users to perform every task required to train and apply the linguistic model to a text this might not be sufficient. As the training and application require knowledge from the domain of computational linguistics, and as different experiments are required to assess the quality of the result, one can hardly claim that end-users are capable to fully exploit the functionality provided.

A similar argument can be raised for the formalization component. As computational modeling is known to be nontrivial, and as it comes along with a high cognitive complexity, the objective to enable untrained users might be very ambitious. The prototype provides a user interface to

create executable models and to define MxL expressions. The knowledge acquisition component is easy-to-use and supports end-users with powerful visualizations to understand the outcome of a reasoning process. However, a structured evaluation regarding the applicability for end-users is up to future work. Based on the feedback of users, their role could be taken into account even further. This would either strengthen the applicability of the overall system within a business environment, or it would generate additional information based on which the system could be improved.

7.2.4. Critical Reflection on the Evaluation

Although a critical reflection on the evaluation of the artifacts within this thesis was already, at least partially, already provided within the evaluation Section 6, this section will briefly summarize the main flaws and limitations of the evaluation as applied.

The case study that was performed with an industry partner within a joint research project mainly focused on the analysis of legal documents to support editorial staff. The project was performed within an interdisciplinary set-up, including the involvement of lawyers. However, only two legal practitioners participated in the project, which is a rather limited number in order to assess the quality of the service that can be provided. The use case was also rather restricted namely the extraction of a particular data value, i.e., year of dispute. To fully assess the performance of the data science framework more diverse use cases need to be implemented and evaluated.

Two main points in the evaluation of the performance with regard to the classification of legal norms could be improved as one could consider them as flaws. The first issue is a conceptual limitation and addresses the classification of semantic types and functional roles on a sub- or super-sentence level, i.e., phrases or paragraphs. Up to now, there is hardly any consensus among the scientific community concerning the right level of annotation to support the interpretation of legal documents. In the evaluation, the assessment has only been performed for sentences within statutory texts.
The second drawback concerns the quantitative analysis of the rule-based and machine learning-based approach. The gold standard for training and testing of the performance of the classifiers was comparably small. Only 532 sentences have been used from a relevant, but also restricted domain: German tenancy law. For more general statements, other legal domains and a larger dataset need to be taken into account.

Finally, although the framework was developed with a strong focus on the end-user, now controlled experiment has been conducted to assess whether the environment actually supports end-users during the interpretation or analysis of legal documents. As already discussed by Figl and Laue (2011), the modeling of business processes is known as a cognitive complex task and requires familiarity with the modeling language and training. The same holds for the formalization of interpreted legal decision structures. This evaluation as performed within thesis, does not allow to make reliable statements on whether the system is supportive during the interpretation task or not.

7.2.5. Critical Reflection on the Research Methodology

The usage of design science, as proposed by Hevner et al. (2004) is very common in information systems research; however some flaws exist. The main and commonly accepted critique certainly is that although it is very concrete in specifying the different phases for information systems research, but very vague in terms of quality criteria for the designed and developed artifacts. Österle et al. (2011) propose a simpler research process consisting of only four different steps, but still, they state the underlying research principle more explicitly, namely abstraction, originality, justification, and benefit. Their proposed method is consistently abstract, as they admit that "Design-oriented IS research is not a non-judgmental scientific discipline, rather it is normative, in a sense that the construction of artifacts is guided by the desire to yield a specific benefit and to satisfy certain objectives." (Österle et al., 2011, p. 3) .

The main advantage of using Hevner's notion of design science is that it constitutes a structured approach to perform the research. In addition, it is constructive and focusing on an artifact appropriate for the work, as described in this thesis. However, it lacks a clear notion of the properties of an artifact and how it can be formalized in such way that it can be assessed accordingly. This problem manifests in during the derivation of the requirements for an artifact, which is mainly done by describing the "business need". However, this concept is rather vague and prevents an exact comparison of different competing artifacts.

Methods to assess the quality of data analysis tasks, as performed in Section 6 and based on Eisenhardt (1989) using standard information retrieval metrics (see Salton 1989), is much more constructive and reliable in terms of reproducibility and expressiveness.

7.3. Outlook

The results and findings from this thesis provide manifold new opportunities for future work. In this section, we focus on and briefly describe four possibilities for future research related to the research prototype that are briefly described. The discussion of future research is centered on the support of (business) processes or relevant use cases.

7.3.1. Legal Text Analytics to Support Business Processes

The prototype was designed and implemented to not only support the analysis of statutory texts from the German legislation, but to allow the integration and analysis of other texts with a particular focus on legal documents, i.e., laws, regulations, statutes, or judgments. A research imposing itself is the extension of the system to more generally support businesses processes more generally, where people work on legal documents in, either by reading, exploring or reviewing them. For example, this is the case in different branches and domains of the industry where compliance is particularly important. Whenever processes and workflows in companies need to consider requirements that are contained in legal documents, e.g., laws, or regulations, the system could help to review them more efficiently and integrate them into the internal process.

This research direction would require a more detailed understanding for the processes of re-

viewing documents for compliance purposes. Based on this investigation, concrete requirements could be derived, focusing on the software supported analysis of these documents. The automated annotation could serve as a decision support tool highlighting the most relevant section within a document or document corpus, with regard to its semantic role.

7.3.2. Legal Text Analytics in Other Domains

Similar to the research direction stated above, a follow-up project could also address the relevancy of legal text analytics in other domains, such as the financial industry or insurance industries. Within these domains, legally relevant documents, such as contracts or terms and conditions, are highly relevant. The content of those documents is primarily unstructured, i.e., text. However, workflows of companies are affected by this content, for example during the claims management process of a company in the insurance industry. Humans need to review the documents received by their clients and compare them to the terms and conditions that apply. A decision support system could determine the most relevant information within the claims document and pre-structure it, so that the review process could be performed more efficiently.

The use cases for a generic framework to semantically analyze legal documents are manifold. However, it is important to structure interdisciplinary legal data science projects along the reference process model as developed and evaluated in this thesis. The differentiation into activities, roles, artifacts, and software-support can be applied to different domains and proactively structures the tasks and their dependencies.

7.3.3. Legal Text Analytics to Support Drafting of Documents

Beside the reviewing and analyzing of legally relevant documents to support processes and workflows, the system could also be adapted to support the drafting of new documents. The main problem this thesis is dealing with, is to extract information from initially unstructured documents. The framework could be adapted to create a new document and already enhance it with information about semantics and structure. This would decrease the effort of extracting information from the document at a later stage.

The framework could be adapted it a way that users receive immediate feedback on the semantic structure of the content that they are writing. This would be applicable to numerous different use cases and scenarios in which legal documents are created. However, the legal document to be drafted needs to be considered within this feedback loop, as the structure and semantics of legal documents differ significantly between different types, such as contracts and laws.

7.3.4. Representation and Modeling of Arguments

The system described a way to support the interpretation of statutory texts and to formalize their computational model in an executable format. Although statutory texts are analyzed on a sentence level, the transition from the text to the executable model may required additional arguments that are not provided in the text. The current implementation does not support the

representation and the modeling of the arguments that lead to a certain decision structure. This highly relevant information remains with the person who created the computational model.

A large community already approaches the representation and formalization of (legal) arguments. Therefore, an interesting and relevant follow-up question would definitely be to address the extension of the interpretation process with respect to codifying the arguments used during the formalization. This would not only increase the transparency of the decision structure but also their comprehensibility.

7.3.5. Formalization of Deontic Concepts, Events, and Actions

Although the system allows to formalize legal decision structures using an expression language that supports propositional, predicate, and higher-order logic, the modeling process itself is still rather static, as each expression defines a derived attribute, which has a pre-defined type, such as Boolean, integer, string, etc. Deontic concepts, such as allowances, or obligations cannot be represented with MxL. Events and actions cannot be modeled by the implementation either. Representing and computationally reasoning on these concepts requires an advanced implementation and additional functionalities of the model and fact store.

It would be interesting to investigate the requirements and the concepts, as well as the impact on the overall implementation of integrating of these deontic concepts. This does not only apply for the software technical implementation, but also for the aspects of modeling, and for applying these for users and for collaborative user interfaces.

Bibliography

Ashley, K. D. (1991). Reasoning with cases and hypotheticals in hypo. *International Journal of Man-Machine Studies*, 34:753–796.

Ashley, K. D. (2002). An ai model of case-based legal argument from a jurisprudential viewpoint. *Artificial Intelligence and Law*, 10:163–218.

Ashley, K. D. (2017). *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*. Cambridge University Press, Cambridge.

Ashley, K. D. and Rissland, E. L. (1988). A case-based approach to modeling legal expertise. *IEEE expert*, 3:70–77.

Bench-Capon, T., Araszkiewicz, M., Ashley, K. D., Atkinson, K., Bex, F., Borges, F., Bourcier, D., Bourgine, P., Conrad, J. G., Francesconi, E., Gordon, T. F., Governatori, G., Leidner, J. L., Lewis, D. D., Loui, R. P., McCarty, T. L., Prakken, H., Schilder, F., Schweighofer, E., Thompson, P., Tyrrell, A., Verheij, B., Walton, D., and Wyner, A. (2012). A history of ai and law in 50 papers: 25 years of the international conference on ai and law. *Artificial Intelligence and Law*.

Bench-Capon, T. and Coenen, F. P. (1992). Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, 1:65 – 86.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific american*, 284:28–37.

Besancon, R., De Chalendar, G., Ferret, O., Gara, F., Mesnard, O., Laïb, M., and Semmar, N. (2010). Lima: A multilingual framework for linguistic analysis and linguistic resources development and evaluation. *Proceedings Conference on Language Resources and Evaluation*, pages 3697 – 3704.

Biagioli, C., Francesconi, E., Passerini, A., Montemagni, S., and Soria, C. (2005). Automatic semantics extraction in law documents. *Proceedings of the International Conference on Artificial Intelligence and Law*, pages 133–140.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Boer, A., Hoekstra, R., and Winkels, R. (2002). Metalex: Legislation in xml. *Proceedings of the Jurix Conference on Legal Knowledge and Information Systems*, pages 1 – 10.

Boitet, C. and Seligman, M. (1994). The whiteboard architecture: A way to integrate heterogeneous components of nlp systems. *Proceedings of the Conference on Computational linguistics*, pages 426–430.

Cardellino, C., Villata, S., Alemany, L. A., and Cabrio, E. (2015). Information extraction with active learning: A case study in legal text. *Proceedings of the International Conference on Computational Linguistics and Intelligent Text Processing*, pages 483–494.

Casellas, N. (2011). *Legal ontology engineering: Methodologies, modelling trends, and the ontology of professional judicial knowledge*. Springer Science and Business Media.

Chalkidis, I., Androutsopoulos, I., and Michos, A. (2017). Extracting contract elements. *Proceedings of the International Conference on Artificial Intelligence and Law*.

Chiticariu, L., Li, Y., and Reiss, F. R. (2013). Rule-based information extraction is dead! long live rule-based information extraction systems! *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 827–832.

Copestake, A. (2007). Semantic composition with (robust) minimal recursion semantics. *Proceedings of the Workshop on Deep Linguistic Processing*, pages 73–80.

Cunningham, H., Maynard, D., and Bontcheva, K. (2011). *Text processing with gate*. Gateway Press CA.

Datev eG (2017). `https://www.datev.com/about-datev/`. Last accessed on April 6th, 2018.

de Castilho, R. E. and Gurevych, I. (2014). A broad-coverage collection of portable nlp components for building shareable analysis pipelines. *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11.

Di Ciccio, C., Marrella, A., and Russo, A. (2015). Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics*, 4(1):29–57.

Dudenredaktion (2013). *Duden Ratgeber - Rechtschreibung und Grammatik*. Bibliograph. Instit. GmbH.

Eisenhardt, K. M. (1989). Building theories from case study research. *The Academy of Management Review*, 14(4):532–550.

Elasticsearch (2017). Documentation. `https://www.elastic.co`. Last accessed on April 6th, 2018.

Ferrucci, D., Lally, A., Verspoor, K., and Nyberg, E. (2009). Unstructured Information Management Architecture (UIMA) Version 1.0. Oasis Standard. *OASIS, Tech. Rep.*

Figl, K. and Laue, R. (2011). Cognitive complexity in business process modeling. *Proceedings of the International Conference on Advanced Information Systems Engineering*, pages 452–466.

Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, Cambridge.

Francesconi, E. (2010). *Semantic processing of legal texts: Where the language of law meets the law of language*. Springer.

Francesconi, E. and Passerini, A. (2007). Automatic classification of provisions in legislative texts. *Artificial Intelligence and Law*, 15:1–17.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Gerathewohl, P. (1987). *Erschließung unbestimmter Rechtsbegriffe mit Hilfe des Computers*. PhD thesis, University of Tübingen.

Glaser, I. (2017). *Semantic Analysis and Structuring of German Legal Documents using Named Entity Recognition and Disambiguation*. Master's thesis, Faculty of Informatics at the Technical University of Munich.

Goncalves, T. and Quaresma, P. (2005). Is linguistic information relevant for the classification of legal texts? *Proceedings of the International Conference on Artificial intelligence and law*, pages 168–176.

Gotz, T. and Suhre, O. (2004). Design and implementation of the uima common analysis system. *IBM Systems Journal*, 43:476–489.

Grabmair, M. (2016). *Modeling Purposive Legal Argumentation and Case Outcome Prediction using Argument Schemes in the Value Judgment Formalism*. PhD thesis, School of Arts and Sciences.

Grabmair, M., Ashley, K. D., Chen, R., Sureshkumar, P., Wang, C., Nyberg, E., and Walker, V. R. (2015). Introducing LUIMA: An Experiment in Legal Conceptual Retrieval of Vaccine Injury Decisions Using a UIMA Type System and Tools. *Proceedings of the International Conference on Artificial intelligence and law*.

Grass, T. (2014). *Development of a Web Application to Manage and Edit Semantically Annotated Texts*. Master's thesis, Faculty of Informatics at the Technical University of Munich.

Gregor, S. (2006). The nature of theory in information systems. *MIS quarterly*, 30:611–642.

Grishman, R. (1996). Tipster text phase 2 architecture design. Technical report, New York University, New York.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Hamp, B. and Feldweg, H. (1997). Germanet - a lexical-semantic net for german. *Proceedings of the Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*.

Hart, H. L. A. and Green, L. (2012). *The concept of law*. Oxford University Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). Overview of supervised learning. *The Elements of Statistical Learning*, pages 9–41.

Hearst, M. (2009). *Search user interfaces*. Cambridge University Press, Cambridge.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *Management Information Systems Quarterly*, 28:75–105.

Hoekstra, R., Breuker, J., Di Bello, M., and Boer, A. (2007). The lkif core ontology of basic legal concepts. *Proceedings of the Conference on Legal Ontologies and Artificial Intelligence Techniques*.

Holmes, G., Donkin, A., and Witten, I. H. (1994). Weka: A machine learning workbench. *Proceedings of the Second Australian and New Zealand Conference on Intelligent Information Systems*, pages 357–361.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47:70–84.

Ide, N. and Pustejovsky, J. (2017). *Handbook of Linguistic Annotation*. Springer.

Jandach, T. (1993). *Juristische Expertensysteme: Methodische Grundlagen ihrer Entwicklung*. Springer-Verlag, Berlin.

Jones, A. J. I. and Sergot, M. (1992). Deontic logic in the representation of law: Towards a methodology. *Artificial Intelligence and Law*, 1(1):45–64.

Jurafsky, D. and Martin, J. H. (2014). *Speech and language processing*, volume 3. Pearson London.

Karpf, J. (1989). Quality assurance of legal expert systems. *International Congress "Logica, Informica, Diritto" Expert Systems in Law*, pages 1–29.

Katakis, I. M., Petasis, G., and Karkaletsis, V. (2016). Clarin-el web-based annotation tool. *Proceedings of the International Conference on Language Resources and Evaluation*.

Katz, D. M. (2012). Quantitative Legal Prediction – or – How I Learned to Stop Worrying and Start Preparing for the Data Driven Future of the Legal Services Industry. *Emory Law Journal*, 62:1–58.

Klügl, P., Töpfer, M., Beck, P.-D., Fette, G., and Puppe, F. (2016). Uima ruta: Rapid development of rule-based information extraction applications. *Natural Language Engineering*, 22:1–40.

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., and Myers, B. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, 43:21.

Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

Krötzsch, M., Simancik, F., and Horrocks, I. (2012). A description logic primer. *arXiv preprint arXiv:1201.4089*.

Larenz, K. and Canaris, C.-W. (1995). *Methodenlehre der Rechtswissenschaft*. Springer, Berlin.

Lei, M., Ge, J., Li, Z., Li, C., Zhou, Y., Zhou, X., and Luo, B. (2017). Automatically classify chinese judgment documents utilizing machine learning algorithms. *Proceedings: International Workshop on Database Systems for Advanced Application*, pages 3–17.

Leith, P. (2010). The rise and fall of the legal expert system. *European Journal of Law and Technology*, 1.

Leon, S. (2001). *Executable Uml: How to Build Class Models*. Prentice Hall PTR.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge university press.

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31.

Maat, E. and Winkels, R. (2007). Categorisation of norms. *Jurix: Conference on Legal Knowledge and Information Systems*, pages 79–88.

Maat, E. d., Krabben, K., and Winkels, R. (2010). Machine learning versus knowledge based classification of legal texts. *Jurix: Conference on Legal Knowledge and Information Systems*, pages 87–96.

Maat, E. d. and Winkels, R. (2010). Automated classification of norms in sources of law. *Proceedings of Workshop on Semantic Processing of Legal Texts*, pages 170–191.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

McCallum, A. K. (2002). Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

Mendes, P. N., Jakob, M., García-Silva, A., and Bizer, C. (2011). Dbpedia spotlight: shedding light on the web of documents. *Proceedings of the 7th international conference on semantic systems*, pages 1–8.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., and Owen, S. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communication of the ACM*, 38(11):39–41.

Modgil, S. and Prakken, H. (2014). The aspic+ framework for structured argumentation: a tutorial. *Argument & Computation*, 5(1):31–62.

Moretti, F. (2013). *Distant reading*. Verso Books.

Mori, S., Nishida, H., and Yamada, H. (1999). *Optical character recognition*. John Wiley & Sons, Inc.

Muhr, J. (2017). *Design, Prototypical Implementation, and Evaluation of an Active Machine Learning Service in the Context of Legal Text Classification*. Master's thesis, Faculty of Informatics of the Technical University of Munich.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30:3–26.

Nay, J. J. (2017). Predicting and understanding law-making with word vectors and an ensemble model. *PLOS ONE*, 12(5).

Neubert, C. (2012). *Facilitating Emergent and Adaptive Information Structures in Enterprise 2.0 Platforms*. Dissertation, Technische Universität München, München.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, pages 841–848.

Nivre, J. (2005). Dependency grammar and dependency parsing. *MSI report*, 5133:1–32.

Nivre, J., Hall, J., and Nilsson, J. (2006). Maltparser: A data-driven parser-generator for dependency parsing. *Proceedings of the International Conference on Language Resources and Evaluation*, 6:2216–2219.

Object Management Group (2011a). Business Process Model and Notation (BPMN), Version 2.0.

Object Management Group (2011b). Unified Modeling Language (UML) 2.4.1 Infrastructure.

Object Management Group (2014). Case Management Model And Notation Version 1.0.

Object Management Group (2015). Decision Model and Notation Version 1.0.

Ogren, P. V. and Bethard, S. J. (2009). Building test suites for uima components. *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 1–4.

Ohlbach, H. J. and Köhler, J. (1999). Modal logics, description logics and arithmetic reasoning. *Artificial Intelligence*, 109:1–31.

Olsson, F. (2009). A literature survey of active machine learning in the context of natural language processing. *SICS Report*.

Oppmann, D. (2016). *Possibilities and Limitations of the Structured Transposition of Normative Texts in Functions on Typed Data Structures*. Master's thesis, Faculty of Informatics of the Technical University of Munich.

Oracle Inc. (2017). `https://www.oracle.com/applications/oracle-policy-automation/index.html`. Last accessed on April 6th, 2018.

Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action*. Manning Publications.

Palmirani, M., Sperberg, R., Vergottini, G., and Vitali, F. (2017). Akoma ntoso part 1: Xml vocabulary. Version 01.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., and Dubourg, V. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77.

Peng, F., Ahmed, N., Li, X., and Lu, Y. (2007). Context sensitive stemming for web search. *Proceedings of the International Conference on Research and Development in Information Retrieval*, pages 639–646.

Petasis, G., Karkaletsis, V., Paliouras, G., Androutsopoulos, I., and Spyropoulos, C. D. (2002). Ellogon: A new text engineering platform. *arXiv preprint*.

Play Framework (2017). Documentation. `https://www.playframework.com/`. Last accessed on April 6th, 2018.

Prakken, H. and Sartor, G. (2015). Law and logic: a review from an argumentation perspective. *Artificial Intelligence*, 227.

redhat Inc. (2017). DROOLS. `http://www.drools.org/`. Last accessed on April 6th, 2018.

Reschenhofer, T. (2013). *Design and Prototypical Implementation of a Model-based Structure for the Definition and Calculation of Enterprise Architecture Key Performance Indicators*. Master's thesis, Faculty of Informatics at the Technical University of Munich.

Reschenhofer, T. (2017). *Empowering End-users to Collaboratively Analyze Evolving Complex Linked Data*. Dissertation, Faculty of Informatics.

Reschenhofer, T., Bhat, M., Hernandez-Mendez, A., and Matthes, F. (2016). Lessons learned in aligning data and model evolution in collaborative information systems. *Proceedings of the International Conference on Software Engineering*.

Reschenhofer, T., Monahov, I., and Matthes, F. (2014). Type-safety in ea model analysis. *IEEE International Enterprise Distributed Object Computing Conference*.

Rissland, E. L. and Skalak, D. B. (1989). Combining case-based and rule-based reasoning: A heuristic approach. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 524–530.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A modern approach*. Pearson.

Sadiq, S. and Governatori, G. (2015). Managing regulatory compliance in business processes. *Spring Handbook on Business Process Management*, 2:265–288.

Salton, G. (1989). Automatic text processing: The transformation, analysis, and retrieval of information by computer. *Addison-Wesley*.

Sartor, G. (1995). *Defeasibility in legal reasoning*, pages 119–157. Springer.

Sartor, G. (2005). *Legal reasoning: A cognitive approach to the law*. Springer, Dordrecht.

Sartor, G., Casanovas, P., Biasiotti, M., and Fernndez-Barrera, M. (2011a). *Approaches to Legal Ontologies: Theories, Domains, Methodologies*. Springer Publishing Company, Incorporated.

Sartor, G., Palmirani, M., Francesconi, E., and Biasiotti, M. A. (2011b). *Legislative XML for the Semantic Web: Principles, Models, Standards for Document Management*. Springer Publishing Company, Inc.

Savelka, J. and Ashley, K. D. (2017). Using conditional random fields to detect different functional types of content in decisions of united states courts with example application to sentence boundary detection. *Workshop on Automated Semantic Analysis of Information in Legal Texts*.

Savelka, J., Trivedi, G., and Ashley, K. D. (2015). Applying an interactive machine learning approach to statutory analysis. *Jurix: International Conference on Legal Knowledge and Information Systems*.

Schaefer, U. (2007). *Integrating deep and shallow natural language processing components: representations and hybrid architectures*. Dissertation, Institute of Informatics.

Sennrich, R., Volk, M., and Schneider, G. (2013). Exploiting synergies between open resources for german dependency parsing, pos-tagging, and morphological analysis. *Conference on Recent Advances in Natural Language Processing*, pages 601–609.

Sergot, M. J., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., and Cory, H. T. (1986). The british nationality act as a logic program. *Communication of the ACM*, 29:370–386.

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, 52:55–66.

Shneiderman, B., Byrd, D., and Croft, W. B. (1997). Clarifying search: A user-interface framework for text searches. *D-lib magazine*, 3(1):18–20.

Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Springer.

Stefanini, M.-H. and Demazeau, Y. (1995). Talisman: A multi-agent system for natural language processing. *Brazilian Symposium on Artificial Intelligence*, pages 312–322.

Sugisaki, K. (2017). Supertagging for domain adaptation: an approach with law texts. *Proceedings of the 16th edition of the International Conference on Articial Intelligence and Law*, pages 249–252.

Surden, H. (2012). Computable contracts. *UC Davis Law Review*, 46.

Susskind, R. (2013). *Tomorrow's Lawyers: An Introduction To Your Future*. Oxford.

Susskind, R. E. (1987). *Expert systems in law: a jurisprudential inquiry*. Clarendon.

Timmer, S. T., Meyer, J.-J. C., Prakken, H., Renooij, S., and Verheij, B. (2015). A structure-guided approach to capturing bayesian reasoning about legal evidence in argumentation. *Proceedings of the International Conference on Artificial Intelligence and Law*.

Walker, V., Hae Han, J., Ni, X., and Yoseda, K. (2017). Semantic types for computational legal reasoning: Propositional connectives and sentence roles in the veterans' claims dataset. *Proceedings of the International Conference on Artificial Intelligence and Law*.

Walter, S. (2010). *Definitionsextraktion aus Urteilstexten*. Dissertation, Saarland University.

Waltl, B., Landthaler, J., Scepankova, E., Matthes, F., Geiger, T., Stocker, C., and Schneider, C. (2017a). Automated extraction of semantic information from german legal documents. *Jusletter IT*.

Waltl, B. and Matthes, F. (2014). Towards measures of complexity: Applying structural and linguistic metrics to german laws. *Jurix: Conference on Legal Knowledge and Information Systems*.

Waltl, B., Matthes, F., Waltl, T., and Grass, T. (2016). Lexia: A data science environment for semantic analysis of german legal texts. *Jusletter IT*.

Waltl, B., Muhr, J., Glaser, I., Bonczek, G., Scepankova, E., and Matthes, F. (2017b). Classifying legal norms with active machine learning. *Jurix: Conference on Legal Knowledge and Information Systems*.

Waltl, B., Reschenhofer, T., and Matthes, F. (2017c). Process and tool-support to collaboratively formalize statutory texts by executable models. *International Conference on Database and Expert Systems Applications*, pages 118–125.

Waltl, B., Zec, M., and Matthes, F. (2015). A data science environment for legal texts. *Jurix: Conference on Legal Knowledge and Information Systems*.

Waltl, T. (2015). *A web based Workbench for Interactive Semantic Text Analysis: Design and Prototypical Implementation*. Master's thesis, Faculty of Informatics at the Technical University of Munich.

Walton, D. (2014). *Abductive reasoning*. University of Alabama Press.

Wilcock, G. (2009). *Introduction to Linguistic Annotation and Text Analytics*. Morgan & Claypool Publishers, San Rafael.

Wilcock, G. (2017). The evolution of text annotation frameworks. *Handbook of Linguistic Annotation*.

Wilson, M. L. (2011). Search user interface design. *Synthesis lectures on information concepts, retrieval, and services*, 3(3):1–143.

Wolinski, F., Vichot, F., and Grémont, O. (1998). Producing nlp-based on-line contentware. *Natural Language Processing and Industrial Applications*.

Wyner, A. (2008). An ontology in owl for legal case-based reasoning. *Artificial Intelligence and Law*, 16(4).

Wyner, A., Mochales-Palau, R., Moens, M.-F., and Milward, D. (2010). Approaches to text mining arguments from legal cases. *Semantic processing of legal texts*.

Wyner, A. and Peters, W. (2010a). Lexical semantics and expert legal knowledge towards the identification of legal case factors. *Jurix: Conference on Legal Knowledge and Information Systems*.

Wyner, A. and Peters, W. (2010b). On rule extraction from regulations. *Jurix: Conference on Legal Knowledge and Information Systems*.

Österle, H., Becker, J., Frank, U., Hess, T., Karagiannis, D., Krcmar, H., Loos, P., Mertens, P., Oberweis, A., and Sinz, E. J. (2011). Memorandum on design-oriented information systems research. *European Journal of Information Systems*, 20(1):7–10.

Abbreviations

AE Analysis Engine

AAE Aggregate Analysis Engine

AML Active Machine Learning

API Application Programming Interface

AUC Area Under the Curve

AST Abstract Syntax Tree

BLOB Binary Large Object

BPMN Business Process Modeling Notation

CAS Common Analysis System

CBR Case-based Reasoning

CMMN Case Management Modeling Notation

CPSL Common Pattern Specification Language

CSV Comma-separated values

DMN Decision Modeling Notation

DSL Domain-Specific Language

DTD Document Type Definition

ES Elasticsearch

FS Feature Selection

FEEL Friendly Enough Expression Language

GATE General Architecture for Text Engineering

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IAAIL International Association for Artificial Intelligence and Law

ISR Information Systems Research

JAPE Java Annotation Patterns Engine

JCAS Java Cover Classes based Object-oriented CAS

JSON JavaScript Object Notation

LES Legal Expert System

LEXIA Legal Information Analysis, Exploration, and Reasoning Platform

LINQ Language Integrated Query

MVC Model View Controller

ML Machine Learning

MxL Model-based Expression Language

NER Named Entity Recognition

NLP Natural Language Processing

OCR Optical Character Recognition

OCL Object Constraint Language

OMG Object Management Group

OPA Oracle Policy Automation

OWL Web Ontology Language

PandF Pipes & Filters

PDF Portable Document Format

RDF Resource Description Framework

REST Representational State Transfer

POS Parts-of-Speech

ROC Receiver Operator Characteristics

RUP Rationale Unified Process

Ruta Rule-based text annotation

Sofa Subject of analysis

UIMA Unstructured Information Management Architecture

UML Unified Modeling Language

XML Extensible Markup Language

YoD Year of Dispute

APPENDIX A

Appendix

A. Requirements Table for the Model-based Reasoning Framework

Import		
1	Flexible import structure	Baseline for the analysis and interpretation is the consideration of different literature (laws, judgments, contracts, commentaries, etc.) that is present in various sources (xml, html, pdf, etc.).
2	Mapping and indexing of legal data	The legal literature has to be indexed and mapped to a data model that does not only preserve the content, i.e. text and metadata, but also structural properties, such as references and nested content.
Analysis		
3	Preserving textual representation	Enabling users to access the content, i.e. legal literature. The visualizations of legal literature have to show the structural information, such as nestedness and links between articles and documents.
4	Collaborative creation and maintenance of patterns	The creation, refinement, and deletion of the required pattern definitions should be done collaboratively in the application, so that different users can share their knowledge and contributions.
5	Lifecycle management of pattern descriptions	Support of the full lifecycle of the pattern specifications, namely creation, refinement, evaluation, and maintenance.
6	Automated pattern detection	Automated identification of linguistic and semantic patterns through data and text mining components.

7	Reuse of existing NLP components	Building of NLP pipelines that allow the easy reuse and sharing of highly specified software components for NLP.
8	Evaluation of annotation quality	Possibility to view the annotations, to examine precision and recall manually, or to export this information to compare against a manually tagged corpus.
9	Manually annotating and commenting of legal texts	Users should be able to manually add relevant semantic information and comments to the legal literature.
10	Storing of annotations	Storing and indexing the automatically determined and manually added annotations.
	Interpretation	
11	Creation of semantic and executable model elements	Stepwise definition of model elements (types, attributes, relationships, operators) for semantic and executable models.
12	Lifecycle support for semantic models	Defining, maintaining, and storing static model elements, such as types, attributes, relationships.
13	Lifecycle support for executable models	Defining, maintaining, and storing executable model elements, such as types, relationships, operators.
14	Correspondence of model elements with text phrases	Creation of connections between model entities and the relevant (interpreted) text. Various levels of the interpreted text should be linkable to model elements, such as words, phrases, sentences, sections, and documents.
15	Domain-specific language (DSL) to express executable semantics	Specification of the operations and executable semantics of relationships with a model-based expression language.
	Application	
16	Access to existing models	Viewing and exploring of semantic and executable models to retrieve the results of prior interpretation processes.
17	Application of models	Executing the defined models through intelligent form-based or spreadsheet-based reasoning.

Table A.1.: Structured requirements to model the semantics of statutory texts.

B. Law Object Mapped into a JSON Document

```
{
    "_index": "default",
    "_type": "Law",
    "_id": "AV9asr7qJdT1sCIZ9dZX",
    "_source": {
        "StartDate": "2013-04-06T03:00:51.000Z",
        "Abbreviation": "ProdHaftG",
        "CreationDate": "2017-10-27T04:59:44.610Z",
        "Language": "de",
        "PromulgationDate": "2013-04-06T03:00:51.000Z",
        "Title": "Gesetz über die Haftung für fehlerhafte Produkte",
    }
}
```

Listing A.1: Simple JSON file for a law object.

C. Implementation of Margin Sampling Query Strategy

```
1  package processing.queryStrategy.uncertaintySampling;
2
3  // imports omitted
4
5  public class MarginSamplingStrategy implements IQueryStrategy {
6
7      @Override
8      public List<Row> getInstancesToLabelNext(List<Dataset<Row>> predictionsList) {
9
10         List<Row> uncertainRows = new LinkedList<>();
11         Dataset<Row> predictions = predictionsList.get(0).cache();
12
13         for(Row row : predictions.collectAsList()){
14
15             int probability = row.fieldIndex("probability");
16             int prediction = row.fieldIndex("prediction");
17             int path = row.fieldIndex("path");
18             int text = row.fieldIndex("text");
19             Vector probVector = (Vector) row.get(probability);
20
21             Double selectedLabel = row.getDouble(prediction);
22             double confidence = probVector.apply(selectedLabel.intValue());
23             double margin = calculateMarginProbability(probVector);
24
25             uncertainRows.add(RowFactory.create(row.get(path),  row.get(prediction)
       ,confidence, margin, row.get(text)));
26         }
27         return uncertainRows;
28     }
29
30     private double calculateMarginProbability(Vector probabilities){
31
32         // get highest probability
33         int maxIndex = probabilities.argmax();
34         double max1 = probabilities.apply(maxIndex);
35
36         // get second highest probability
37         double[] probabilitiesArray = probabilities.toArray();
38         probabilitiesArray[maxIndex] = 0.0;
39
40         int maxIndex2 = probabilities.argmax();
41         double max2 = probabilities.apply(maxIndex2);
42
43         return max1 - max2;
44     }
45 }
```

Listing A.2: Implementation of the query strategy “margin sampling strategy”.