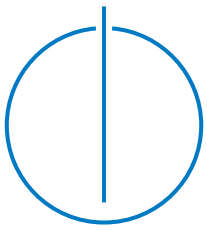
Department of Informatics
Technische Universität München

Master's Thesis in Informatics

Possibilities and Limitations of the Structured Transposition of Normative Texts in Functions on Typed Data Structures

Dominik Oppmann



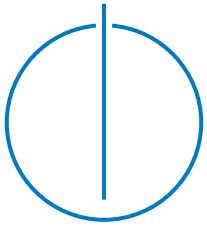
Department of Informatics
Technische Universität München

Master's Thesis in Informatics

Möglichkeiten und Grenzen der systematischen Übersetzung juristischer Texte in Funktionen über typisierten Datenstrukturen

Possibilities and Limitations of the Structured Transposition of Normative Texts in Functions on Typed Data Structures

Author:	Dominik Oppmann
Supervisor:	Prof. Dr. rer.nat. Florian Matthes
Advisor:	Bernhard Waltl, M.Sc., M.A.
Submission Date:	15.10.2016



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15.10.2016

Dominik Oppmann

Zusammenfassung

Normative Bestimmungen, beschrieben durch Gesetze, Dekrete und Urteile, sind hauptsächlich in textueller Form dargestellt. Diese Darstellungsart hat den Nachteil, dass sie eine einfache Verständlichkeit misst und dass viele unklare Informationen enthalten sind, die zuerst interpretiert werden müssen. Der Prozess der Normeninterpretation ist allerdings sehr komplex auf Grund des hohen Zeit- und Arbeitsumfangs und dem Mangel an unterstützenden Software-Tools. Deswegen trägt diese Masterarbeit zur Formalisierung für Modelle basierend auf Rechtstexten durch die Verwendung eines geeigneten Tools bei.

Das Hauptziel dieser Arbeit ist die Erstellung eines graphischen Modelleditors, mit dem Rechtsexperten die Entscheidungsstrukturen in Rechtstexten formalisieren und somit die Semantiken in einem graphischen Modell festhalten können. Der beabsichtigte Formalisierungsprozess sieht vor, dass Nutzer verschiedene Dokumente zu einem Modell-Workspace hinzufügen und aus diesen ein "semantisches Modell" entwickeln. Dieses Modell besteht aus Typen mit Attributen und Beziehungen zwischen diesen Typen. Alle Modellelemente können zu den importierten Texten verlinkt werden, um ihren Ursprung zu rechtfertigen. Die eigentlichen Entscheidungsstrukturen sind in den Attributen modelliert, die neben primitiven Datentypen auch ausführbare Ausdrücke (MxL) beinhalten können.

Der zweite Teil der Arbeit besteht aus der Entwicklung einer Evaluierungsumgebung für semantische Modelle, um diese mit Daten zu befüllen und die ausführbaren Ausdrücke zur Laufzeit auf diesen Daten auswerten zu können. Deswegen wird SocioCortex als Inferenzmaschine eingesetzt. Diese Evaluierungsumgebung ermöglicht auch unerfahrenen Nutzern, die nicht im Detail mit den Entscheidungsstrukturen gewisser Normen vertraut sind, diese zur Simulation von (hypothetischen) Fällen zu nutzen.

Abstract

Normative regulations, described in laws, decrees and judgments, are mainly represented in textual form. But this representation lacks of an easy comprehensibility and it also contains a lot of vague information that must be interpreted first. The whole task of norm interpretation is very complex, due to its time and labor intensity and the lack of proper tool support. Therefore, this master's thesis contributes to formalization for models based on legal texts by the usage of proper tool support.

The main goal in this work is the creation of a graphical model editor, which helps legal experts formalizing the decision-making structures in normative texts and represent the semantics of a text in a visual model. In the intended formalization process, the user can use several legal documents as base for creating a "semantic model" consisting of types with attributes and relations between each other. All model elements can be linked to the source text to justify their origin. The decisions are modeled in the attributes which can hold next to primitive data types also executable expressions (MxL).

The second part of this thesis consists of the creation of an environment to populate these semantic models with concrete data instances and evaluating the MxL expressions at runtime. Therefore, SocioCortex is used as a reasoning engine. The benefit of the evaluation environment is to enable the simulation for (hypothetical) cases even by unexperienced user that do not understand the intended semantics of normative texts in detail.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

In civil law countries such as Germany, the most important legal sources are laws, which are produced by the parliamentary system. Additional sources such as government decrees, ministerial decrees and court decisions must be considered for legislation as well, which is the base of all public services provided for citizens or businesses. Therefore, understanding the intended semantics of this legislation is a crucial task. It is also the first step towards a transposition into a more formal specification. In a next step, these specifications can be implemented in a computer executable way allowing an automatic processing.

In the past, main attention was paid on this second step, namely the automatic execution of law. However, the previously mentioned upstream transposition process from rules described in natural language into a computer executable formal model was neglected. Up until now, the translation process is mainly driven by legal experts, such as lawyers or legal data scientists, during an *interpretation* task. This task puts focus on understanding how a norm should be applied [11], [38]. Interpretation is the preliminary stage for *subsumption* which checks whether facts of a case are applicable for some given norms [22, p. 83ff].

Moreover, the *interpretation* process of norms is a very complex task due to several reasons. At first, this process is very data intensive. It can be the case that norms regarding the same circumstances are distributed over several acts. Brattinga et al. [11] identified this phenomena for the past Dutch environmental law, which was spread across 26 different acts. Van Engers et al. [38] refer to this as "scoping issue", not knowing where to start analyzing the law and where to stop. A second issue is that the law contains a huge amount

of implicit and vague information that must also be clarified and made explicit first [37].

These two problems are attributable to the main representation form of legal norms, which is unstructured text. Since text provides an "excellent job in creating legal security, along with all the legal authorities" [11] this representation form will still be relevant in the future. Hence, concepts were developed during the last 25 years around these textual representations to support the comprehensibility of its semantics. As an example *annotations* [20] must be stated, which were originally used to enrich cases with meta data. Its purpose is to provide cover terms for legal concepts which should homogenize different linguistic patterns expressing semantically the same. Moreover, a concept called *isomorphism* has proved to be beneficial. It refers to a linkage between a formal model of some legal semantics and their textual origins [9].

Based up on different formalization methods of normative texts, a broad variety of *legal expert systems* were developed during the last two centuries. These systems should advise legal experts during their daily work and help at different levels. Typical tasks of such systems are decision support, training, intelligent databases for searching and text analysis systems [23, p. 10 ff.]. Especially the latter ones gain popularity through recent artificial intelligence and machine learning techniques.

The chair of *Software Engineering for Business Information Systems* at the TU München has been developing *Lexia*, which is a legal data science environment and analysis platform. One of its main features is the analysis of legal texts regarding linguistic patterns with the goal to identify legal concepts in a huge text corpus. This task is achieved through the usage of natural language processing technologies. Currently, this platform lacks of an component that supports legal experts during interpretation of legal norms and enforcing the creation of a formal model as resulting artifact on base of the previously identified legal concepts. These formal models should be executable such that unexperienced and untrained users can use them for the evaluation of cases.

This thesis describes the creation of such a component including a previous literature research which helped revealing requirements for this component. After the implementation, an evaluation has been realized to verify whether certain legal semantics can be implemented with this component in a formal model. Furthermore, the intended solution separates the task of *interpretation*

and *subsumption*, that was previously done only by a single person, to foster the reuse of these formal models.

1.2 A brief history of AI and law

The usage of techniques of artificial intelligence in the domain of law is quite a new discipline. First researches were done in the 80s of the last century. A real community around this topic has begun to grow with the *International Conference on AI and Law* (ICAIL) in 1987. Ever since, the conference takes place on a biennially base presenting the latest achievements on this field of research [10, p. 2f].

One important milestone of AI in law was presented even in the early days of the ICAIL conference. In 1987 Hafner [10, p. 6ff] showed the limitations of normal string search in legal texts and presented "concept indexation and concept search" as a possible solution. In normal boolean string search, expert knowledge is required in order to identify relevant search terms and their combinations. With the usage of *annotations*, relevant text passages could be indexed by cover terms. These terms are "semantically meaningful generalizations" that eliminate homogeneous text passages expressing semantically the same circumstances. Searching for these annotations would lead to better results with respect to accuracy rate and number of search terms used. Documents must not only be annotated, but also a mapping between annotations and relevant legal concepts must be defined [10, p. 7]. Hafner used annotations for cases represented in textual form. Since then it is widely used for instance in case-based reasoning systems as part of the information retrieval process [6]. Still today annotations play a major role in some legal expert systems.

Annotating the legal documents manually is a time- and labor-intensive task. Since the beginning of AI and Law, attention has been paid on *information retrieval* (IR), which is the search for words, sentences and larger phrases in a huge amount of datasets. First attempts were made in the 1980s, which were mainly rule-based. Key challenges at that time were not only technical issues such as the user interface, coverage of the text corpora and response time, but also - from a legal point of view - the usage of many different phrases expressing the same idea. The latter problem with the synonyms and polysemes has

been addressed with different kinds of thesauruses, e.g. a " norm based thesaurus". The idea behind this was a unified and normalized presentation of the document structure [10, p. 10-11]. Later the idea got extended and instead of a thesaurus, lexicons were used which contain "legal concepts, cited cases, cited statutes, proper names and facts". Even advanced information retrieval, neural networks and machine-learning tools could not create a sufficiently unified formal representation automatically from legal texts [10, p. 60-61], which makes it still a manual, complex and time-consuming task.

However, a formalized representation of legal documents is a prerequisite in order to build legal expert systems with automatic reasoning. During the last decades, many legal expert system approaches have been introduced. As examples HYPO (Rissland and Ashley, 1987), CABARET (Rissland and Skalak, 1989) and CATO (Aleven, 1997) are to be named. Since the late 1980s, the case-based reasoning systems were one of the main research subjects of artificial intelligence and law. Another subject, which has been in focus of research since the 1990s, are rule-based approaches for reasoning. Main contributors are Gordon (1995), Prakken and Sartor (1996) and Hage (1997) [10, p. 32ff]. Other approaches, for instance neural networks, have led to partly satisfying results. Section 2.3 gives an introduction on legal expert systems and also contains a short overview of selected legal expert systems.

The introduction should give insights into the developments of AI and law during the last 25 years. The achievements made in the early years, such as information retrieval techniques and annotations, are still relevant and in use today. Lexia (3.1.1) has such features implemented to analyze texts regarding their semantic and linguistic structure and mark them with annotations. An alternative representation of the semantics in legal texts, represented in a formal model, still cannot be done automatically and consequently it remains a manual task. Hence, this thesis addresses this issue and describes how Lexia is enhanced to support legal experts with the creation of formal models.

1.3 Objectives

The objective of this master's thesis is the development of a prototypical modeling environment, which enables different users to create and also to evaluate

formal models out of normative texts. This modeling environment is implemented in Lexia, a legal data science environment developed at the chair "Software Engineering for Business Information Systems" of the TU München. In two case studies, which exemplary have different legal concepts and decision-making structures, the modeling environment is evaluated.

The following six research questions have been derived and will be answered concretely and in detail in Section 5.1:

1. What are the possibilities of formalizing semantics of normative texts?
2. Which components are required to enhance Lexia for fulfilling the task of model formalization and evaluation?
3. How could an approach look like to link elements of a formal model with its textual representations?
4. How would a concrete formalization of two selected case studies (child benefit of EStG and reporting obligation of BDSG) in the newly implemented modeling environment of Lexia look like?
5. What would be a suitable meta model for the representation of formalized semantics in SocioCortex?
6. What are the benefits of separating the task of creating a formal model and applying it?

1.4 Structure

In Chapter 1, a short motivation and a brief history of the recent work in the field of AI and law is given, as well as the objectives and the structure of this thesis are introduced. The next chapter (2) deals with related work and should make the reader familiar with previous approaches of formal models, as well as legal expert systems. Chapter 3 proposes the modeling approach and presents some details of the involved systems. Furthermore, the case studies and the requirement analysis are described, as well as the stakeholders. In the implementation section (4), the implemented solution is shown in detail and the changes made in the front end as well as in the back end are described. Subsequently in an evaluation (5), the research questions defined at the beginning are answered and the implemented models of the two case studies are

elucidated. The thesis concludes with a short summary and an overview of recent trends in decision modeling (6).

2 Related work

The related work chapter of this master's thesis elaborates the benefits and requirements of formal models (see Section 2.1), knowledge representations in the domain of law (see Section 2.2) and legal expert systems (see Section 2.3).

2.1 Formalization

This section describes at first the benefits of using a model compared to the traditional text form (see Section 2.1.1). Then requirements for such formal models are collected through literature research (see Section 2.1.2), based on former formalization approaches.

2.1.1 Benefits of formalization

At the beginning (see Section 1.1) it was mentioned that the main representation of legal norms are in unstructured, textual form. This form has several drawbacks, such as lacking of easy comprehensibility, because of the quite unusual used phrased and linguistic patterns. Moreover, the huge amount of implicit information contained in such documents contributes to the poor comprehensibility. Therefore, laymen are not able to fully capture the semantics of complex legal decision-making structures. As a proposed solution, building a *model* as alternative representation that makes some abstraction could solve these issues.

In general, a model is an image of reality, which is limited only to several aspects. This limitation of selected properties of a system is crucial [34, p. 78f], [17]. The intended solution in this thesis is a *semantic model*, limited to the representation of legal entities with attributes and their relationship in between (see Section 3.3). This kind of model only highlights the static

aspects. Moreover, it can be extended by a rule based expression language that also captures dynamic aspects and makes the model executable.

With the help of models the comprehensibility is improved, because it resolves previously mentioned issues about the representation in textual form. Moreover, vague or implicit assumptions must be made explicit first in order to use them in such a model. The task of model creation is typically fulfilled by legal experts, that have a deep understanding of the domain. As described in the section about the history of AI and law (see Section 1.2), the generation of such models can even with the latest tools not be done fully automatically.

Once a *semantic model* has been created, even unexperienced users can work with it by applying facts to these models and simulate potential outcomes. This is exactly the task of *subsumption* in jurisprudence. A proper formal model is therefore the foundation of legal reasoning, which is usually the task of *legal expert systems* (see Section 2.3). With an explicit formal model, the reuse of such semantic models is also promoted.

In summary, the benefits of a formal model compared to the unstructured text is a better comprehensibility as well as the ability to execute the defined semantics in such models. Once a formal model was created, it can be reused for simulating different cases.

2.1.2 Requirements to formal models

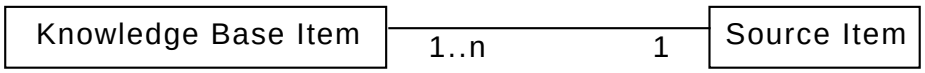
At the beginning of the formalization approaches in the 1980s, different considerations were undertaken, which led to several important aspects that are still applicable today.

[10, p. 8] argue that a model should be usable for different stakeholders, who have various requirements for the model. For example, clerks who want to apply the law do not have the same requirements as solicitors who advise how the law is to be applied. However, creating different representations of the same model (or using the same legal text as base) could lead to redundancy and inconsistency. The authors instead suggest building a logical and executable formalized model, which can be used as foundation for further enhancements. Multi-user-models have also been considered by van Engers' latest research

[37]. He mentions that a formalized model must " [...] [represent] norms in ways, that enable multiple task contexts and multiple agents perspectives".

Bench-Capon et al. [10] also propose the idea that there must be a link between the entities in the model and the original legal text in order to provide transparency. Their later research lead them to define the term *isomorphism* as a "well defined correspondence of the knowledge base to the source texts", addressing the concerns about comprehensibility, maintainability and validation [9]. They suggest that ideally there must be a one-to-one relation between items from the knowledge base and source items. Due to practical reasons, this condition does not hold and must be relaxed such that one source item relates to several knowledge base items (see Figure 2.1). The cardinality of the inverse relation between knowledge base item and source item must be limited to one.

Figure 2.1: The mapping between *Knowledge Base Item* and *Source Item* suggested by Bench-Capon et al.



Source: Own illustration, based on description in [9]

Another argument of the necessity of such a linkage is provided by van Engers et al. [15] which refers to the example of Web- and IT-Services, provided by authorities. They point out that nearly every service, provided by public administration and other governmental institutions, are primarily based on laws or other regulations. In order to handle these public services effectively, they are implemented and deployed as IT-Services. However, in most cases the connection between IT-Service and legal source is not clear which is problematic in terms of maintenance and system complexity. Brattinga et al. [11] specify this connection even further. They argue that the link must be a two-way-link, because for reasons of justification and validation. The task of formalization is typically done by knowledge analysts and knowledge engineers. In enterprise settings, this is handled in a collaborative process with groups of analysts or engineers with normal professional skills. However, the actual validation of the formal model is on the other hand done by legal and policy experts. In

order to validate the completeness and soundness of these models, a transparent translation process must be provided as well as a link between model and source [37].

Early formalization approaches were mainly driven by legal experts as primary source of knowledge. As described in [36], for the formalization of the *British Nationality Act* from 1986 (see Section 2.2.1), an expert was the main source of legal knowledge. During the years this paradigm has changed and now experts are only used as controllers and interpreters of the models for knowledge representation [36], [37]. Nowadays the aim is to use a systematic transposition process, which uses computational linguistics approaches to firstly identify relevant patterns in the text and secondly transpose these text fragments in model elements. At the time writing this thesis, this process must still be done manually, because of the lack of proper tool support [34, p. 94].

A further challenge, that must be addressed while formalizing legal texts, is the problem of scoping. Laws contain implicit and explicit references to other paragraphs within the legal document as well as to other documents, forming a network of relationships. However, these relationships only depend on the current context of the case and are not applicable for all cases. These circumstances make it hard for an analyst to define where to start investigating and when to stop looking for additional sources [38].

Besides that, the comprehensibility of the resulting model is an important aspect. If a model is not comprehensible for legal experts and other administrative workers, for example if it is expressed very formal in rational algebra, it is not usable in practice [37].

Summarizing all previous requirements for formal models results in the need of a link between the source text and the formal model for traceability and validation reasons. Ideally the formal model is understandable by normal users and exists in any kind of executable form. Furthermore, different perspectives for different stakeholders are needed. The process of creation and refinement of such models must be designed in a collaborative way and must address the issue of scoping. The main stakeholders creating and maintaining these models will be legal experts, because of a lack of tool support for automatic knowledge retrieval.

2.2 Models for knowledge representation in the domain of law

The previous Section 2.1.1 gives an introduction regarding formal models and why it is beneficial to use such models. This section is exemplary showing different approaches of representing knowledge in the legal domain. The rule-based approach (see Section 2.2.1), the case-based approach (see Section 2.2.2), the logic-based approach (see Section 2.2.3) and ontologies (see Section 2.2.4) are elucidated. *Knowledge representation* is tightly connected with *legal expert systems* that are using a formalized representation for legal reasoning (see Section 2.3).

2.2.1 Rule-based approach

One of the first approaches of representing knowledge of a legal document was in sets of rules. This form of representation simplifies legal reasoning in rule-based legal expert systems (for an example see Section 2.3.3.1) and is based on the assumption that law can be formalized with so called production rules [34, p. 85].

In the following, the example by Sergot [36] is used, who formalized the *British Nationality Act* in Horn clauses. That was a necessary prerequisite for the later implementation in logic based programming languages such as Prolog. Horn clauses are logical sentences, which consist of disjunctions with at most one positive literal. This form allows to rewrite the sentence as a implication with a conjunction of positive literals as premise and a single positive literal as conclusion [33, p. 256f].

The Listing 2.1 shows the first paragraph of the *British Nationality Act* from 1981 in natural language:

Listing 2.1: First paragraph of the BNA in natural language

```
A person born in the United Kingdom after com-
mencement shall be a British citizen if at the time
of birth his father or mother is
   (a) a British citizen; or
   (b) settled in the United Kingdom.
```

In comparison to this, Listing 2.2 shows a formalized version in Horn clauses which can be directly converted to Prolog source code.

Listing 2.2: First paragraph of the BNA in Horn clause form

```
x is a British citizen
  if x was born in the U.K.
  and x was born on date y
  and y is after or on commencement
  and z is a parent of x
  and z is a British citizen on date y
```

Of course this formalization approach is done on a very low abstraction level (directly in executable code). Other approaches on a much higher abstraction level are also possible. Section 2.3.3.1 describes the *Oracle Policy Automation*, which is a system that enables the modeling and deployment of business rules. The approach introduced in this thesis is also based on rules (see Section 3.3) on a higher level of abstraction.

An advantage of this approach is that one would have an executable representation of the semantics very quickly with no intermediate formalization steps. In comparison with the requirements from Section 2.1.2, this model does not provide any transparency on how it was generated. Furthermore, only skilled experts, which can read source code, can use these kinds of models for application and validation purpose. A further drawback is the lack of linking between legal text and source code which affects the maintainability. If the legal text is changed, these changes cannot be tracked to the relevant parts of the source code easily or even automatically. Changes in legal texts must be cumbersomely implemented in code again. This process is very cost-intensive as well as error-prone.

Scharf [34, p. 87] observes that law has a hierarchical structure and is mainly composed of legal definitions which are hard to represent with rules. Rules are not adequate as single form of modeling, but are useful for representing parts of the knowledge.

2.2.2 Case-based approach

This section about case-based approaches is only sketched briefly due to reasons of completeness, but has limited relevance for the further work.

The case-based approach is based on the assumption that the total amount of legal knowledge is only collected in cases. This stands in contrast to the rule based approach in which it is assumed that law can be expressed in rules. Therefore, the case-based approach is only valid in the anglo-american common law system [34, p. 87f].

The second assumption is that cases could be judged by comparing them to similar cases, because it is likely that similar cases will have a similar outcome [34, p. 88f], [27, p. 39]. Hence, the cases must be prepared in such a way that comparison between them is easily achievable. This can be fulfilled through the previously mentioned concept of *annotations* (see Section 1.2).

The representation of the knowledge encoded in cases and the processing of it depends highly on the legal expert system which works with this knowledge. Therefore, a general representation does not exist and cannot be given. In the section about legal expert systems a closer look is given to *HYPO*, developed by Rissland et al. [32]. It is also elucidated how the *case knowledge base* is organized for that system (see Section 2.3.3.3).

2.2.3 Logic-based approach

The following section gives an summary over different types of logic-based knowledge representation approaches, such as *deontic logics* (2.2.3.1), *non-monotonic logics* (2.2.3.2) and *description logics* (2.2.3.3).

2.2.3.1 Deontic logics

Deontic logics refers to deontic terms such as *obligation* and *permission*. Scharf [34] notes that deontic logic is used for formalization of law, because the law itself uses the deontic terms. Furthermore, it is used to represent the differences of what should have been and what actually has been [34, p. 90].

One early approach of using deontic logics for formalizing the law was used by Hohfeld [21] in 1919. His work was not actually a formalization, but Hohfeld led a solid foundation for further usage. Hohfeld introduced *jural relations*,

which always appear as pairs. Table 2.1 shows these jural relations, e.g. *right* is a jural opposite of *duty*.[1]

Table 2.1: Jural relations by Hohfeld

Jural opposites			
right (claim)	privilege (liberty)	power	immunity
duty	no-right	liability	disability

Source: [21, p. 30]

There have been several approaches for a more precise formalization based on Hohfelds work. One of the latest contributions were done by van Engers et al. [15] and Brattinga et al. [11]. They used the original Hohfeld model, but extended it with temporal aspects and also with the ability to create new legal relations.

Van Engers et al. categorize the jural relations in two categories: Category A, which enables to create or destroy jural relations, including themself (POWER-LIABILITY and the opposite DISABILITY-IMMUNITY) and category B, which are used for describing pre- and postconditions for category A relations (CLAIMRIGHT-DUTY and the opposite NORIGHT-LIBERTY). For instance, the NORIGHT-LIBERTY pair expresses that somebody (Person A) has a *liberty* on subject matter M and a second Person (Person B) has *no right* to interfere the liberty of Person A [37].

As concrete example, van Engers et al. use regulations from the Dutch Immigration and Naturalization Service (IND) [37], in which a foreign student applies for permanent residence in the Netherlands:

> "The foreign student that wants to study in the Netherlands, comes to the IND for legal residence. Article 8, Aliens Act declares that having a residence permit gives the alien the LIBERTY to have legal residence, which leaves the other, passive party in the jural relation based on this article with a NO RIGHT."
>
> "Our Minister is not mentioned in article 8, but there is an explicit reference to article 14, which gives Our Minister the POWER to

[1] Hohfeld also defined jural correlatives, but these are not used in this context and therefore not explained.

> grant, to reject, or disregard the application for a residence permit. The alien is the explicit actor holding a LIBERTY in article 8, and the implicit patient holding a LIABILTY in article 14."

The quoted case above shows how the pattern of identifying these jural relations is applied to a case. These jural relations can then be transformed into a formal model using specific relational algebra (e.g. van Engers et al. use a specific application of CogNIAM [38]).

2.2.3.2 Non-monotonic logics

Non-monotonic logics are used to build rules with exception. In comparison to monotonic logic, a conclusion which was drawn once by a set of premises can never be invalidated. Adding new facts to the set of premises does not change a conclusion. However, non-monotonic logic follows more the common sense logic: Conclusions are drawn based on assumptions of the world, which we expect to be *normal*. This behavior is best if dealing with incomplete information. It can happen that an assumption, which previously was expected to be normal, is wrong because of new information. In this case, the drawn conclusion must be adjustable [39].

This is the way non-monotonic logics behave, namely a conclusion can be made invalid through the addition of new premises. Non-monotonic logics describe the normal or *default case*, but also allow for exceptions in some special cases. This property of non-monotonic logics can be used to model the fact that law is build on rules that can also contain exceptions [34, p. 92].

2.2.3.3 Description logics

"Description logics (DLs) [...] are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way." [7]. With DL, important notions of a domain can be depicted by descriptions, which are expressions consisting of atomic concepts (unary predicates) and atomic roles (binary predicates). In addition to that, description logics can express formal, logic-based semantics which was not possible with their predecessors, such as semantic networks. Furthermore, they are a decidable subset of first-order logics.

DL supports a terminological ($TBox$) and assertional ($ABox$) formalism which together form the total knowledge base of a domain. Terminological axioms are used to express the domain knowledge and their relationships, whereas assertional axioms contain concrete instances of the domain [34, p. 91]. Equation 2.1 shows the definition of a Tbox expression ("only humans can have human children") and equation 2.2 depicts an example of an assertional formalism ("Mary is one of Bobs children")[2].

$$\exists hasChild.Human \sqsubseteq Human \tag{2.1}$$

$$hasChild(BOB, MARY) \tag{2.2}$$

According to Baader, Horrocks and Sattler [7], description logics are the perfect candidates for ontology languages, because they have well-defined semantics and can also be used for reasoning. The explanation of ontologies is continued in Section 2.2.4

Logics provide a strict formalism to express the semantics of normative regulations. Different types of logic try to purge drawbacks of other types, e.g. the *non-monotonic logics* enable reverting a drawn conclusion, if new premises are added to the sets of facts and the *deontic logics* allow the modeling of obligations and permissions. *Description logics* provide a decidable subset of first-order logic and is widely used for ontologies.

The problem with logic-based approaches in general is their decidability. It does not make sense to use any kind of higher-order logic, which is not decidable and consequently an inference machine cannot deduce conclusions. Hence, the usage of logics is a trade-off between complexity and expressiveness [29, p. 280f].

Another drawback in logics is the lack of expressing arithmetical operations. In several norms, such as the in Section 3.4.1 presented *Einkommenssteuergesetz*, basic arithmetic operations have to be performed in order to calculate the child benefit. Baader and Sattler identified these problems in [8] for DLs and also introduced an extension to add the basic aggregation functions *min*, *max*,

[2] These examples are taken from [7] and are slightly modified.

count and *sum*. However, these basic functions do not provide the full variety like a formalization in a programming language would.

2.2.4 Ontologies

The term ontology has its origin from philosophy, but was later defined by Gruber [19] as "an explicit specification of a conceptualization", meaning that "a conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose."

An ontology, used in terms of knowledge representation in artificial intelligence, uses a reusable vocabulary to express phenomena of the real world in a computer understandable way. Furthermore, ontologies extend taxonomies, in which concepts are depicted as classes that can be enriched with attributes, axioms, restrictions and relations to other classes [34, p. 109].

In order to express ontologies, different ontology languages were developed. In the following, the *Web Ontology Language 2 (OWL2)* is exemplarily used. For representing the actual ontology, *OWL 2* uses *RDF* (Resource Description Framework), consisting of triplets of subject, predicate and object, for knowledge representation. With RDF, it is possible to create a knowledge representation for arbitrary domains [34, p. 105f]. RDF documents can also be expressed in different syntaxes, for instance XML or Turtle. However, the XML syntax is the standard one, which must be supported by all OWL2 tools [3], [5], [4].

The two Listings 2.3 and 2.4 show the different syntaxes, where the first one is in XML and the latter one in Turtle. These examples are taken from [5] and [4].

Listing 2.3: Example of RDF in XML syntax

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
       dc:title ="RDF/XML Syntax Specification (Revised)">
    <ex:editor>
      <rdf:Description ex:fullName="Dave Beckett">
  <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.4: Example of RDF in Turtle syntax

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

Apart from the syntax, OWL2 uses logics to describe complex information which is used by inference machines for reasoning. OWL2 supports the in Section 2.2.3 explained description logics. Regarding their versatility, ontologies have also been used in the legal domain. Several approaches have been undertaken so far to formalize knowledge of the legal domain with ontologies [29], [34]. Ontologies using description logics suffer from the same drawbacks as description logics, namely the poor expressiveness of arithmetic operations.

2.3 Legal expert systems

This section should give a short introduction what (legal) expert systems are and what they consist of. Then, exemplary legal expert systems are described such as the *Oracle Policy Automation* (2.3.3.1), *knowledgeTools* (2.3.3.2) and *HYPO* (2.3.3.3).

2.3.1 Definition

The definition of an expert system is not very precise and unified. One definition describes it as a computer program, which solves problems, that are at least so hard that normally human expert knowledge is required. For problem solving, knowledge and inference techniques are used [23, p. 6]. Puppe enhanced this definition by adding the constraint that these systems can only operate in a very limited range [28, p. 2]. In order to build an expert system, knowledge must be previously *formalized* and *represented* in the system (see Section 2.1 and 2.2).

Therefore, legal expert systems are expert systems that are used in the legal domain and also represent knowledge and inference rules of this domain.

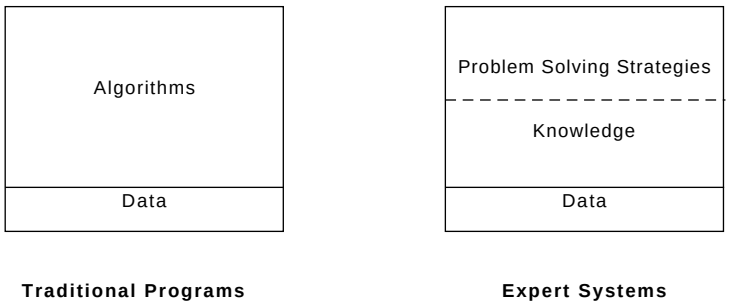
2.3.2 Architecture

A main aspect of the architecture of expert systems is the separation of *knowledge* and *problem solving strategies* [28, p. 2]. Figure 2.2 shows the distinction between classical programs and knowledge-based systems. In traditional programs, the algorithms are fixed and cannot be changed at runtime, whereas in knowledge-based systems the problem solving strategies are dynamic and can be extended with the *knowledge acquisition facility*.

A classical expert system consists of the following components [23], [28]:

Knowledge Base contains the knowledge of the system. This knowledge can be separated into *facts*, *rules* and *meta-rules* describing the usage of rules.

Figure 2.2: Comparison of traditional programs and knowledge-based systems



Source: Own illustration, based on [28, p. 2]

Inference Engine uses the expert knowledge in the knowledge base to solve problems.

Explanation Component helps to visualize and to justify the procedure of how the expert system gained new knowledge. This can either be used for experts to evaluate the knowledge base system as well as for an unexperienced user to get an explanation for a solution.

Knowledge Acquisition Component enables the domain experts to enter knowledge into the knowledge base of a system and also to modify it later.

Dialog Component is used to process the user input and respond with outputs for the user.

2.3.3 Examples of legal expert systems

During the last 25 years, several legal expert systems were developed that support legal experts during several tasks. In this section, three different systems are exemplarily elucidated. At first, *Oracle Policy Automation* (Section 2.3.3.1) and *knowledgeTools* (Section 2.3.3.2) as representative of rule- and logic-based legal expert systems. Techniques implemented in these systems

were partly adopted for the approach presented in this thesis (Section 3.3). At the end, a very influential and most cited legal expert system *HYPO* is shown that is a representative of a case-based reasoning system (Section 2.3.3.3).

2.3.3.1 Oracle Policy Automation

The *Oracle Policy Automation* (OPA) documentation [2] claims the design purpose of the system as to "[...] deliver consistent and auditable advice across channels and business processes by capturing rules in natural language Microsoft Word and Excel documents and building interactive customer service experiences called interviews around those rules."

OPA is a software suite consisting of several components such as a desktop modeling tool for policy experts, a management console for the deployment of policies as well as a web-based form (called interview) to advice customer for desktop and mobile clients.

Figure 2.3 shows an example of rules in natural language that are written with Microsoft Word. These rules can then compiled with the *Rule Assistant* provided by OPA and used for example as eligibility checks or for calculations. Rules have to be composed in an *if-then-manner* starting with the conclusion first and followed by the conditions.

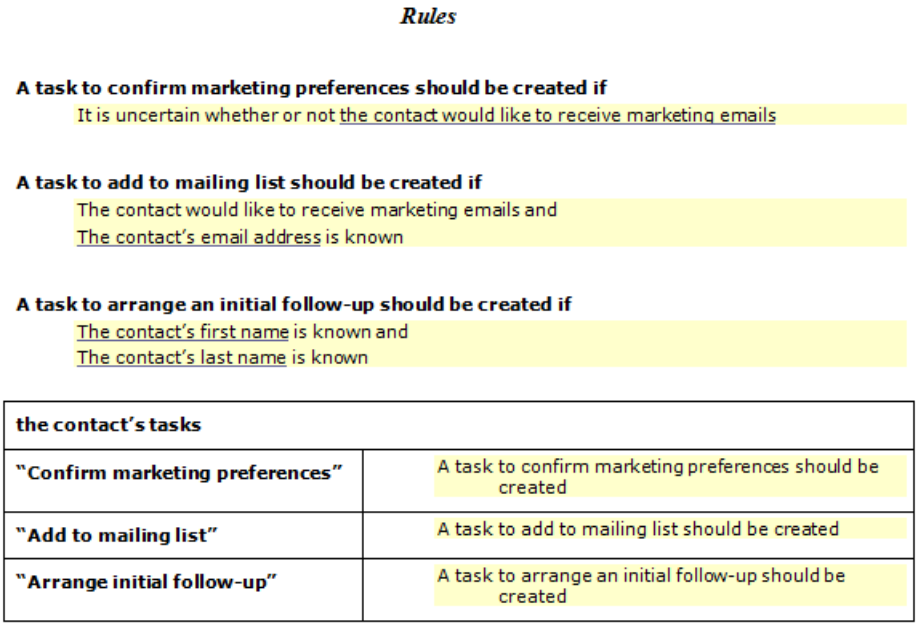
The policy modeling tool expects that the opposite conclusion can be inferred if the conditions are not satisfied. Besides *boolean rules* (boolean attribute is inferred from conditions), also *assignment rules* (a value is assigned to an attribute) and *table rules* (attribute value is inferred from multiple rules depending on the situation) are supported. The rules can contain the boolean operators *and* and *or*, as well as grouping operators such as *both*, *all*, *either* and *any*. An example for the usage of grouping operators is shown in Listing 2.5.[3] Also the standard logical operators and numerical functions can be used for rule modeling.

[3] This example is taken from the Oracle Policy Automation documentation: http://documentation.custhelp.com/euf/assets/devdocs/august2016/PolicyAutomation/en/Default.htm#Guides/Policy_Modeling_User_Guide/Rule_writing/Logical_connectors_in_rules.htm

Listing 2.5: Example of a rules for *Oracle Policy Automation*

```
the claimant is eligible for a pension if
 the claimant is poor or
  all
    the claimant is sick and
    the claimant has been sick for more than 6 months and
    the claimant does not have another form of income
 or
 the claimant has been entitled to a pension previously
```

Figure 2.3: The rules in natural language in Microsoft Word



Rules

A task to confirm marketing preferences should be created if

It is uncertain whether or not the contact would like to receive marketing emails

A task to add to mailing list should be created if

The contact would like to receive marketing emails and
The contact's email address is known

A task to arrange an initial follow-up should be created if

The contact's first name is known and
The contact's last name is known

the contact's tasks	
"Confirm marketing preferences"	A task to confirm marketing preferences should be created
"Add to mailing list"	A task to add to mailing list should be created
"Arrange initial follow-up"	A task to arrange an initial follow-up should be created

Source: Policy Modeling User Guide[4]

The *Oracle Policy Automation* provides a system for unexperienced business users to model semantics described in natural language and enable interactive reasoning through a web interface.

2.3.3.2 knowledgeTools

knowledgeTools is a software suite from knowledgeTools International GmbH, which consists of several modules. Especially the *case management module* is

[4] http://documentation.custhelp.com/euf/assets/devdocs/august2016/PolicyAutomation/en/Content/Guides/Policy_Modeling_User_Guide/Getting_started/What_is_Oracle_Policy_Modeling.htm

interesting for decision modeling, because it is a database oriented system allowing the visualization of complex legal decision-making structures. A similar approach is also the objective of this thesis.

Figure 2.4 shows a screenshot from the screencast provided on the knowledgeTools website [1]. Breidenbach describes in [12] the structure and the functionality of this system: The knowledge is organized in a *knowledge tree* (dt. Wissensbaum) that consists of several individual nodes. The nodes follow a hierarchical composition principle, starting with the actual question at the most left side of the tree and connected by decision nodes that are either atomic nodes or contain further sub nodes.

The relations between the individual nodes are either logical AND, OR, or XOR (exclusive or). Nodes can be evaluated to *true* (highlighted with green at the left side of a decision node) or *false* (highlighted with red). After the decisions have been made, the knowledge tree propagates these information to nodes on higher levels to perform a backtracking up until the root node which contains the question.

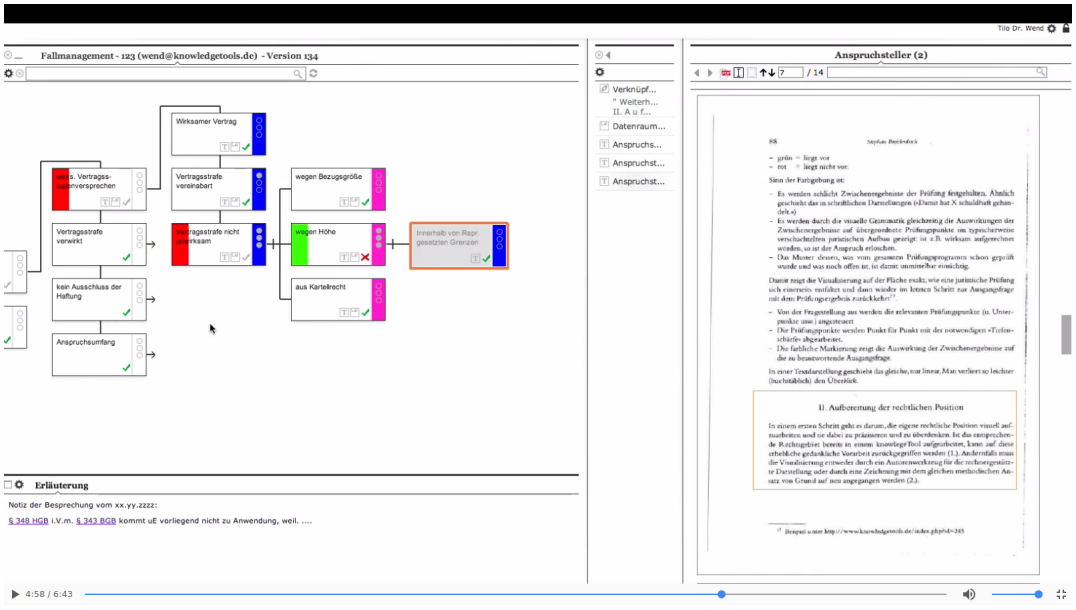
Each node can be linked with several documents that are displayed on the right side of the screen (see Figure 2.4). These documents are indexed in a full text search engine. Besides the documents, text fragments can be assigned to each node allowing to automatically generate a claim document regarding the modeled case and the evaluated decisions. Through a message board system custom comments can be drafted and assigned to each node.

In [35], a typical use case is presented from the domain of social law which is supported by the knowledgeTools system. It allows a lawyer to check together with his client whether a received decree from an authority is valid. The lawyer has a model of the case in the system and asks the client several facts about the circumstances. Next, these facts are entered into the system and a contradicting conclusion is drawn. knowledgeTools has automatically generated a claim document for this client which only must be customized slightly.

2.3.3.3 HYPO

HYPO is a case-based (see Section 2.2.2) reasoning system that is used in the domain of trade law. It operates on a set of given facts to perform a legal analysis and finally presents the conclusion in form of argument outlines

Figure 2.4: knowledgeTools case management component



Source: knowledgeTools screencast[1]

with references to previous cases. It does not actually "decide" cases, rather than provides arguments on behalf of the involved parties. Field of usage for example is the assistance of attorneys [32], [27, p. 39].

HYPOs knowledge base is separated into two repositories, the *Case Knowledge Base* (CKB) and the *library of dimensions*. The CKB stores all cases, regardless of whether they are real or hypothetical, organized as hierarchical frames whose slots are relevant features of the cases (such as the plaintiff or defendant).[5] The library of dimensions provides several features for classification of cases [32].

If all current facts (*or the current fact situation cfs*) have been entered into HYPO, it starts with its legal analysis process and generates a *case-analysis-record*, which identifies the dimensions that apply for the entered case and which are almost applied. This combined list is then called *D-List*. Next, on base of the in the previous step generated D-List, the user is prompted to input

[5] The terms *frames* and *slots* refer to a knowledge representation framework, introduced by Minsky [25]. *Frames* are "[...] data-structure[s] for representing a stereotyped situation [...]". Minsky uses the example of going to a child's birthday party as an instance of a *frame*. With a frame, several information can be attached to it. Frames also have *slots*, which can be interpreted as attributes and "[...] must be filled by specific instances or data". Frames are organized in frame-systems that consists of several frames organized in a hierarchical way.

additional information, which HYPO uses for example for legal reasoning. With the case-analysis report and the additional information, HYPO generates a *claim-lattice*, which as root node displays the D-List and the current facts. The child nodes are other cases from HYPOs knowledge base, whose D-Lists are subsets of the current fact situation. With these techniques, most-on-point cases can be identified, as well as least-on-point cases. With the claim-lattice and the most-on-point cases for the plaintiff and defendant side, cases are selected on which to rely or to distinguish for an argument. The claim-lattice is also used in HYPO for the generation of hypothetical cases for the case knowledge base. In a last step, the constructed argument skeleton is expanded to provide justification and explanation.

3 Analysis & concepts

This chapter explains at first two systems involved in this thesis, in order to understand the general modeling concept which relies on these systems (see Section 3.1). Before the actual modeling approach is introduced in Section 3.3, the model based expression language MxL is presented (see Section 3.2). Furthermore, two case studies, the child benefit from the *Einkommenssteuergesetz* as well as the reporting obligation from the *Bundesdatenschutzgesetz* are introduced in Section 3.4, before the actual stakeholders (see Section 3.5) and the requirements of this stakeholders (see Section 3.6) are addressed.

3.1 Description of the involved systems

3.1.1 Lexia

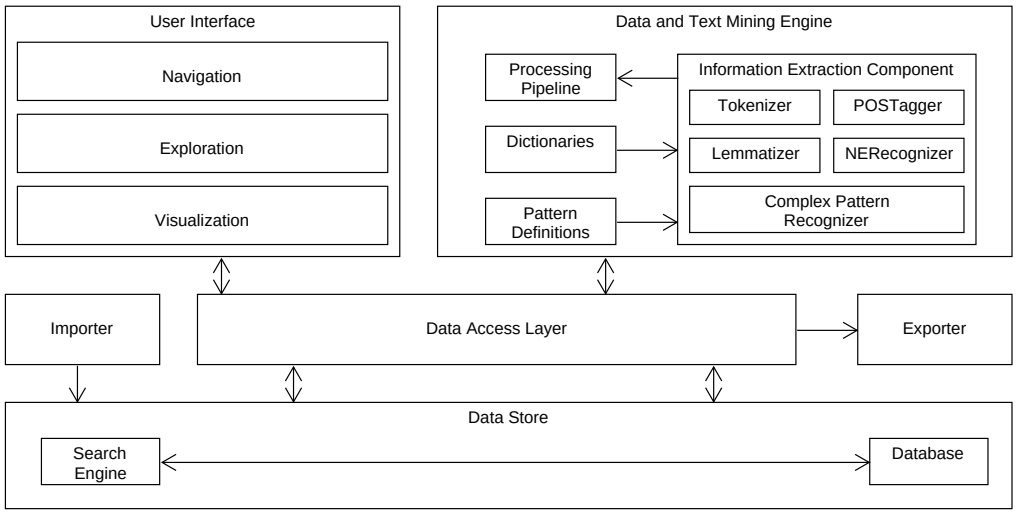
Lexia[6] is a web based "data science environment for semantic analysis of German legal texts" [41] with the goal to analyze legal texts from different sources regarding their linguistic structure. It has been developed at the chair of "Software Engineering for Business Information Systems" (sebis) at the TU München and is a research approach of tailoring generic *Natural Language Processing* (NLP) components to the domain of legal data science. This tailoring process is necessary in order to achieve "highest accuracy in terms of prediction and recall" [41].

The development of Lexia was inspired by the latest developments in computer science (especially in the field of artificial intelligence) and should support handling of legal tasks, which are in general very data-, time-, and knowledge intensive [41].

[6] Further information for Lexia respectively for the Lexalyze Interdisciplinary Research Program can be found at https://wwwmatthes.in.tum.de/pages/12bpn04x6h3x8/Lexalyze-Interdisciplinary-Research-Program

The architecture is based on the *Apache UIMA* (Unstructured Information Management Architecture) reference architecture, developed by IBM and also used in IBM Watson. With the rule language *Apache RUTA* (Rule-based Text Annotation), patterns can be expressed in an easily maintainable and reusable way. The architecture of Lexia is depicted in Figure 3.1. It consists of several components such as the importer or exporter for importing legal documents with different file formats (e.g. PDF, Word file) or exporting data dumps of semantic entities over a REST API. The text analysis engine, following a pipes-and-filter-architecture, provides state of the art methods (e.g. Splitter, Segmenter, Tokenizer, Tagger, Ruta) for analyzing the structure of the text. As storage and search engine, the schemaless *ElasticSearch* is used [41].

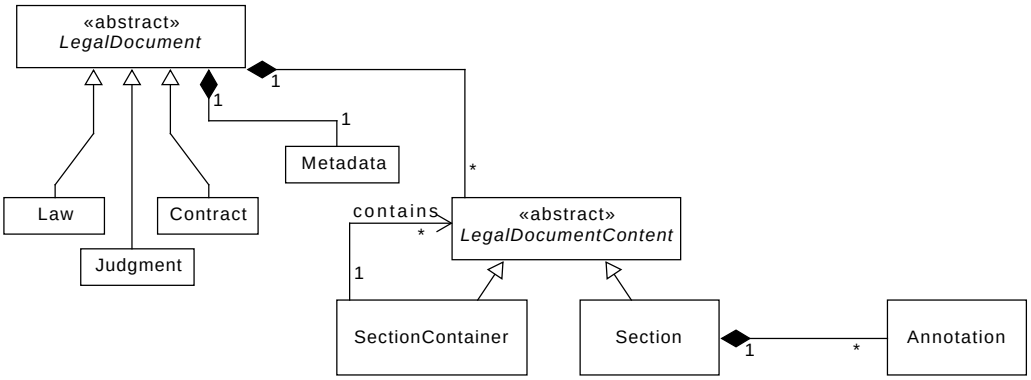
Figure 3.1: Architecture of Lexia



Source: [41]

The special data model, tailored to the structure of legal documents (see Figure 3.2), makes it easy for extension. The abstract base class of all document types is *LegalDocument*, from which all specializations are inherited. Each *LegalDocument* has associated *Metadata* that contains additional information. In order to represent the nested structure of legal documents, a composite pattern was chosen for the *LegalDocumentContent*. The actual textual content is then stored in *Sections* which can be enriched with *Annotations*.

Figure 3.2: Data model of Lexia



Source: [41]

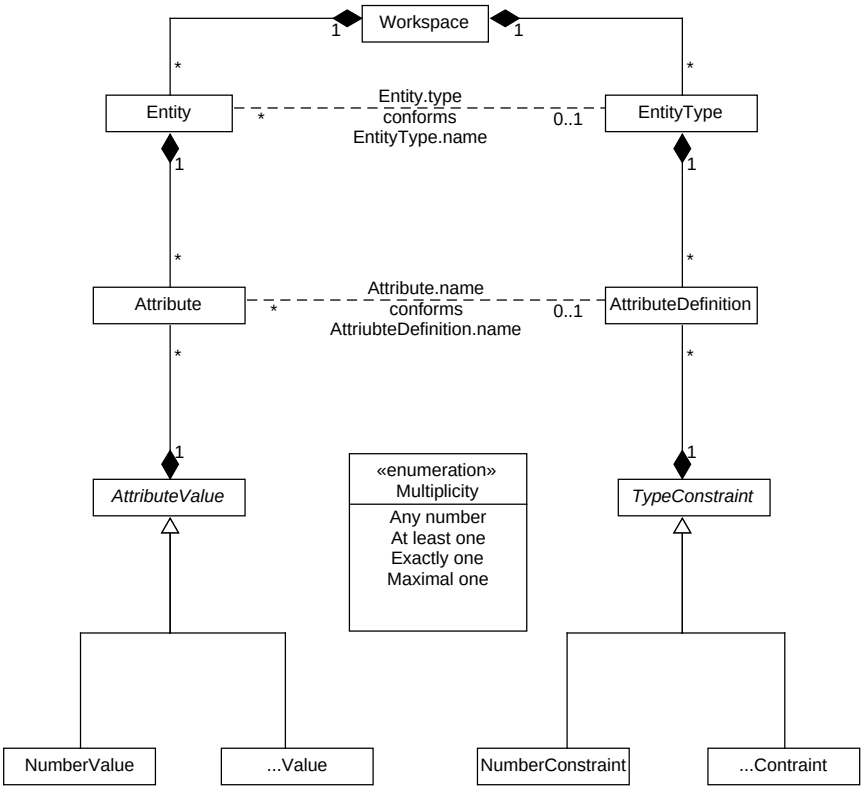
3.1.2 SocioCortex

SocioCortex (formerly known as Tricia) is a hybrid wiki system that is also developed by the sebis chair. The hybrid wiki concept was first introduced by Matthes et al. in [24] with the goal to create a lightweight and structured data management system on top of an unstructured wiki system. With this approach it should be avoided that users have to learn a new and complex semantic language for structuring wiki pages. This hybrid wiki is designed to either create the data model first (top down) or to import data first (bottom up) and then guide the users towards a consistent and formally defined data model through the addition of constraints.

This approach was evaluated in several projects [31] and an adaption has been made which led to the following meta model shown in Figure 3.3.

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

Figure 3.3: Meta model of SocioCortex



Source: Own illustration, based on [31]

SocioCortex is also equipped with a powerful REST API[7] which is heavily used in this thesis (see Section 4.2.1.5). With this REST API all elements of the meta model can be created, queried, modified and deleted. JSON (JavaScript Object Notation) is used as data exchange format.

3.2 The model based expression language MxL

MxL, the *model based expression language* respectively its predecessor Txl (Tricia Expression Language) was introduced by Monahov et al. in [26]. It is a domain specific language for the definition and computation of key performance indicators (KPI) in the domain of *Enterprise Architecture Management* (EAM). Its reference implementation was realized in the EAM tool Tricia (now SocioCortex, see Section 3.1.2) which follows a model based wiki approach

[7] Documentation is available at http://www.sociocortex.com/documentation/

named *hybrid wiki*. To TxL's core features count queries on the EA model, including aggregation of data as well as arithmetical and logical operations.

TxL is a functional language which supports object orientation and it has a dynamic type system with reflection. Dynamic binding of functions at runtime promotes their reusability. The language can operate with simple data types like *Object, String, Number, Boolean* and *Date*, as well as constructor data types like *Sequence, Map, Function* and *Entity*. As arithmetic operators, *addition, subtraction, multiplication, division, integer division* and *modulo* are implemented. For boolean comparisons, *isNull, equals, greaterThan* and *lessThanOrEqualsTo* are present as well as the three basic logical operators *and, or, not*.

To achieve projection and selection functionalities, TxL supports since the beginning different query (e.g *select, where, groupBy*) and aggregation (eg. *count, sum, min, max*) operators. These operators are inspired by *Microsoft's standard query operators* [8] [9].

A major update of TxL and a simultaneously renaming to MxL was done by Reschenhofer [30]. In his master's thesis, he identified the shortcomings of the first prototypical implementation of TxL. The missing compile-time analysis of TxL expressions was a major drawback. If the underlying EA model had been changed and types had been removed or altered, it would have made the expressions invalid, resulting in a runtime exception. Moreover, the tight coupling between TxL and Tricia turned out to be a drawback, because this domain specific language (DSL) had the capability to be used also in other tools than Tricia.

Reschenhofer addressed these above mentioned issues and also introduced new data types (*Page, Document, Principal, Person* and *Group*), changed arithmetic, logical and comparison operators from functions to symbolic terms (e.g. from *1.0.add(2.0)* to *1.0 + 2.0*) and also made minor syntactical modifications (e.g. for conditionals). Furthermore, he added also the concept of *derived attributes*. This attribute types for Tricia are not persisted, but instead computed at runtime. All of these changes result in version 2.0 of MxL.

[8] see https://msdn.microsoft.com/en-us/library/bb394939.aspx

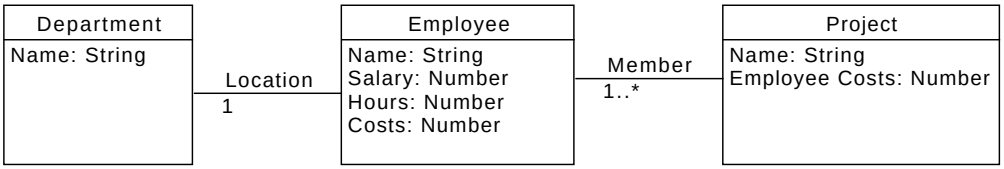
[9] The complete list of supported query and aggregation operators is shown in Table II and III in [26]

Listing 3.1 shows an MxL 2.0 expression that queries all departments which contain at least on employee with an salary greater than 20. The corresponding data model is shown in Figure 3.4.

Listing 3.1: Example of an MxL query

```
find(Department)
  .where(d => d.get Employee whereis Location
    .any(e => e.Salary > 20 ))
```

Figure 3.4: Data model for MxL expression in Listing 3.1



Source: Own illustration, based on MxL tutorial[10]

3.3 Modeling approach for Lexia

This thesis uses the previously introduced hybrid wiki *SocioCortex* (see Section 3.1.2) and the expression language *MxL* (see Section 3.2) for a formalization approach. Formal models (see Section 2.2), in the following referred to *semantic models*, are expressed as sets of typed data structures with relations between each other. Further refinement can be achieved with attributes for these data structures that can hold either primitive data values (such as *String* or *Number*) or custom MxL expressions (to which is now referred as *derived attributes*). These derived attributes are then evaluated once a formal model has been created in SocioCortex and populated with data. In Section 3.3.3, the meta model for the semantic model is depicted, describing the individual entities. The whole formalization approach can be seen as a *rule-based approach* which is influenced by *ontologies* in terms of knowledge representation (see Section 2.2.1 for rule-based knowledge representation and 2.2.4 for ontologies).

[10] see http://www.sociocortex.com/tutorial/2015/12/01/mxl05/

For a convenient creation of these semantic models, a visual model editor is provided in Lexia. This editor enables the generation of a visual representation for a *semantic model* which increases the comprehensibility. Therefore, a main contribution of this master's thesis is the implementation of such a model editor. Visual support is not only provided during the creation of semantic models, but also for the evaluation as well. More concretely this means during the process of entering data and evaluating the derived attributes. Section 3.3.1 and 3.3.2 show the workflow of how models are created and persisted in SocioCortex, as well as how these models are instantiated with concrete data instances.

A separation of these workflows was intended in order to provide two different user interfaces for the two main stakeholders. The stakeholders are closer analyzed in Section 3.5. As mentioned at the beginning, it is tried to separate the *interpretation* process of legal norms from their *subsumption* (see Section 1.1).

SocioCortex together with MxL were chosen in this formalization approach, because SocioCortex is a representative of a system with a dynamic meta model (see Section 3.1.2). The meta model can be tailored specifically matching the needs to represent the semantic model. Hence a mapping must be created which maps elements of the semantic model with entities in SocioCortex. This mapping is presented in Section 4.1.2.

Furthermore, SocioCortex has been developed by a research group at the same chair where this master's thesis is set up. In case of any issues the developers are physical available and can provide support.

3.3.1 Workflow of model building

Figure 3.5 describes a typical workflow of how a legal data scientist would create a *semantic model*.

The user opens Lexia and creates a new semantic model, which is initially empty and immediately saved in Lexia. After that, he opens the newly created semantic model and adds documents to it, which are relevant for modeling a certain aspect described in these documents. Next, based on the documents, the user starts to add new types, links them with textual sources and also

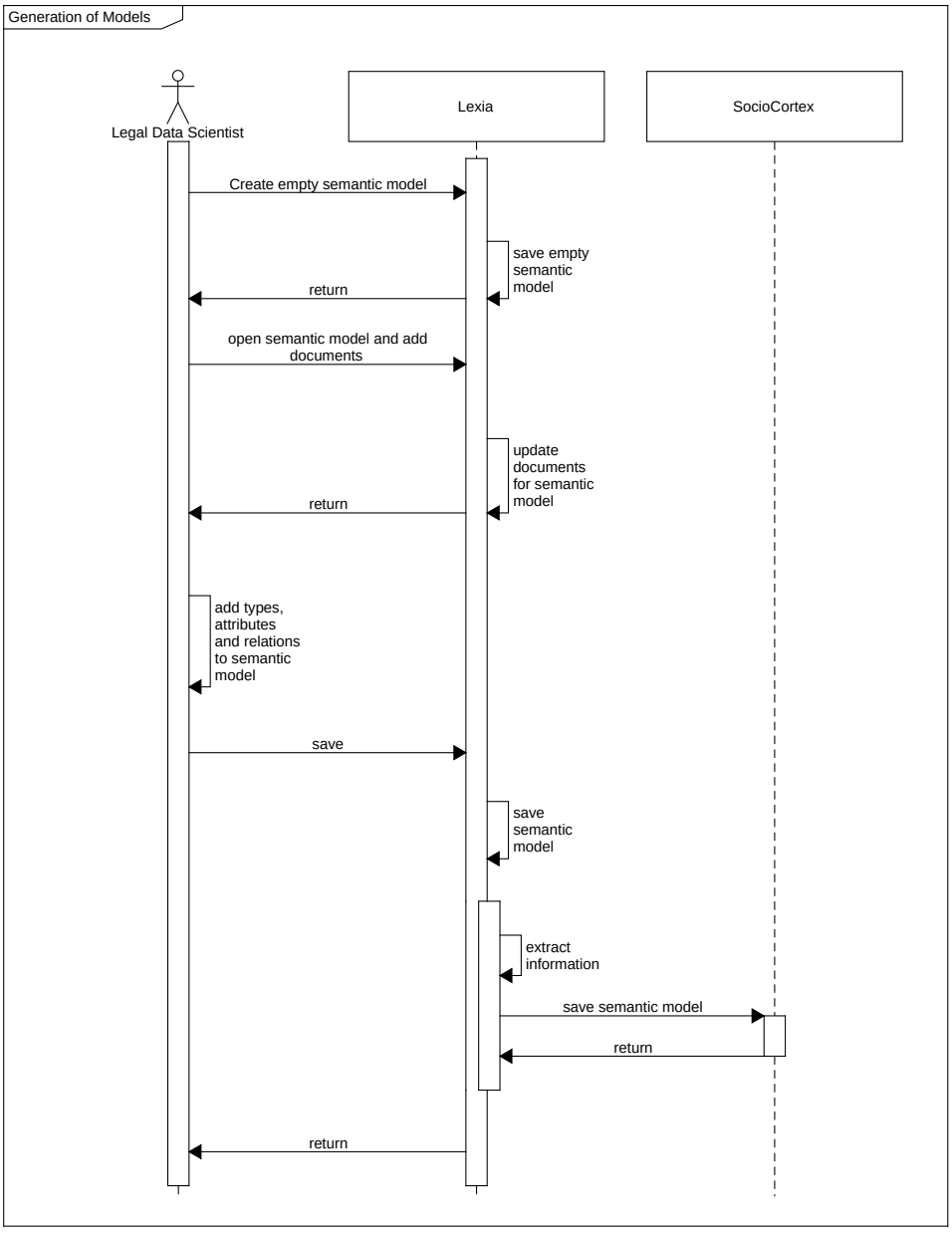
refines those types with attributes and relations. After the modeling is finished, the complete visual representation of this model is saved in Lexia and informations are extracted from these visual model: All types, attributes and relations are extracted and created in SocioCortex, too. Therefore, a mapping schema must exist between semantic model elements and SocioCortex entities (see Section 4.1.2). A first version of the created semantic model is now done and can either be refined by adding or removing new types, attributes and relations or it can be evaluated. The workflow of evaluating a model is described in the next section (see Section 3.3.2).

3.3.2 Workflow of model evaluation

After a model has been created, it can be populated with data and also be evaluated. Figure 3.6 shows the process of evaluating a model.

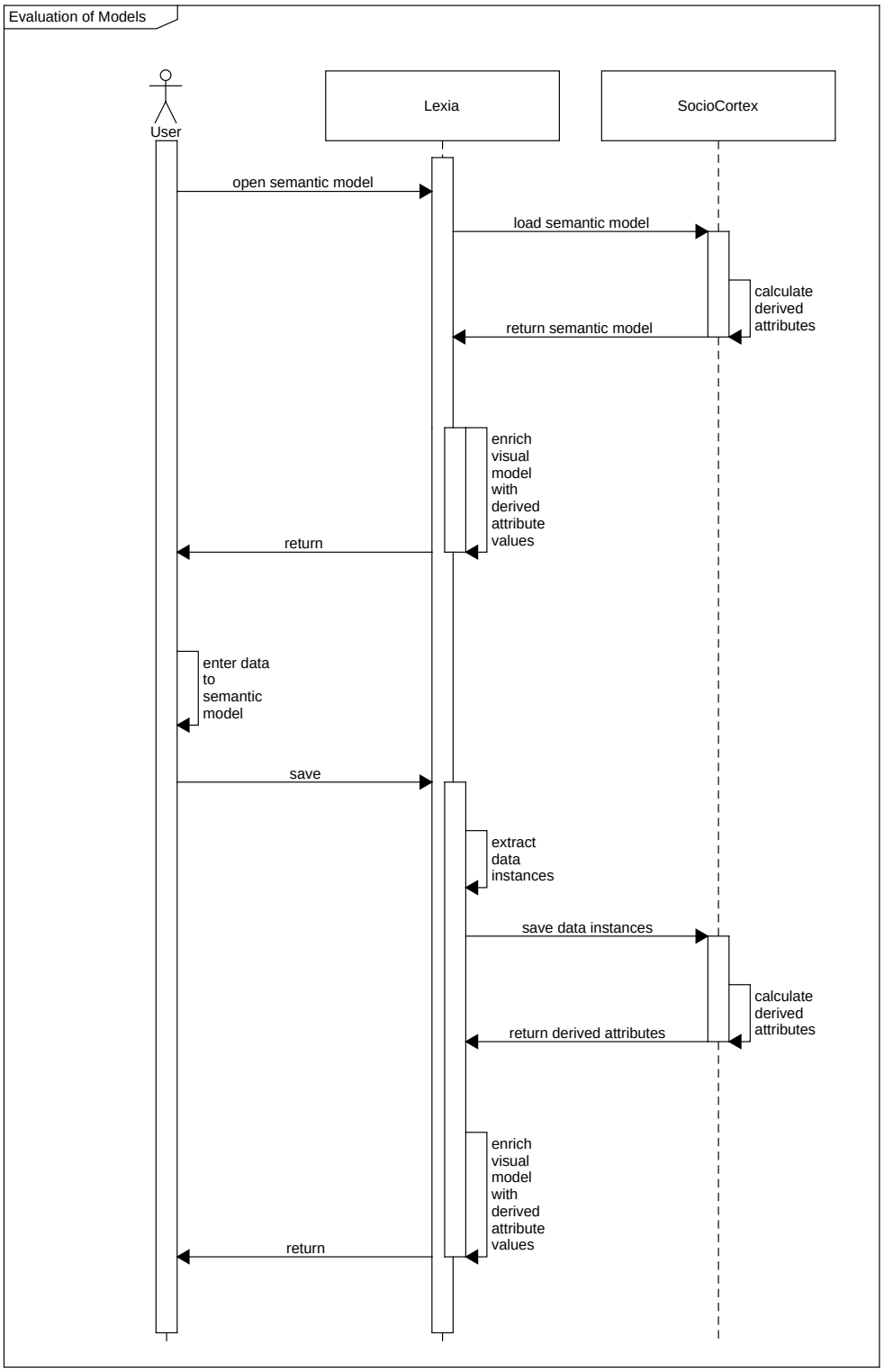
At first, a user opens a previously created model in the evaluation view. At that time, the semantic model along with the newly calculated derived attributes are fetched from SocioCortex. If the user has not entered data before, the model is empty and otherwise the derived attribute values from SocioCortex are inserted in the visual model. The whole model is returned subsequently to the user and rendered on his screen. The user has now the possibility to modify data in a form or to enter new data instances. After saving the changed or newly created instances, the data is first extracted from the visual model and then saved in SocioCortex. Furthermore, a recalculation of the derived attributes is triggered and returned to Lexia, which enriches the visual model with the newly evaluated values. After that, the complete visual model is returned and displayed in the user's browser again.

Figure 3.5: Workflow of the generation of semantic models



Source: Own illustration

Figure 3.6: Workflow of the evaluation of semantic models



Source: Own illustration

3.3.3 Meta model of semantic models

Figure 3.7 shows the meta model for the *semantic model*. In contrast to the presented formalism of Bench-Capon in Section 2.1.2 and visualized in Figure 2.1, the association between a *source item* and a *model element*[11] is a many-to-many relationship, allowing a model element to be linked with more than one source item. The weakening of this constraint were undertaken, because in practice more than one source is needed to explain a legal construct. Despite our civil law system in which jurisdiction is based on laws, in practice preceding cases and other literature such as table of fees are also relevant. With the loosed condition, multiple text sources can be linked to a model element.

A model element can be either a *type*, a *relation* or an *attribute*. A relation can only be established between two different types and a type can have more than one relation. Between two types only one relation can be established. Furthermore, a type can have attributes and each attribute belongs only to one type. An attribute cannot exist on its own.

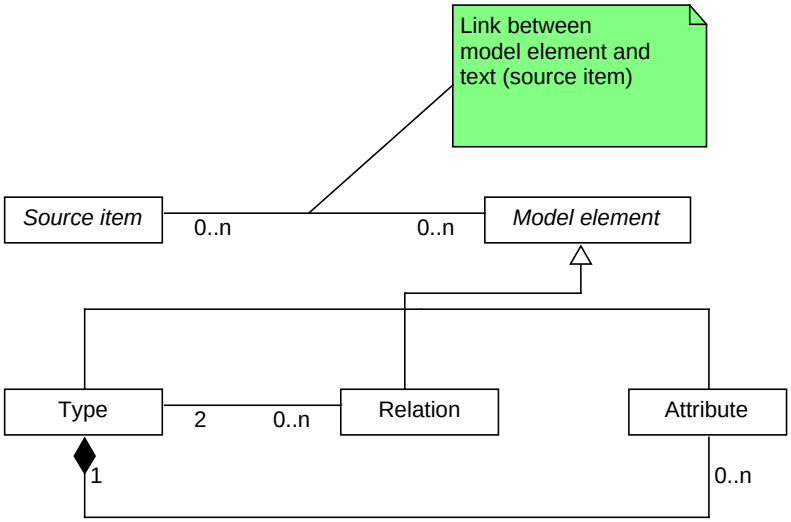
3.4 Case studies

In a case study two different laws are investigated regarding their possibilities to model their semantics (partly) with the approach introduced in this thesis. The child benefit calculation described in *Einkommensteuergesetz* (EStG) and the reporting obligation to the data privacy authority (Datenschutzaufsichtsbehörde) described in *Bundesdatenschutzgesetz* (BDSG) were chosen. These two norms have been selected, because it is supposed that the semantics of the norms can be modeled and the dynamic aspects be described with MxL (see Section 3.2).

In Section 5.2 the results of the formalization of these two case studies are demonstrated.

[11] In the terminology of Bench-Capon a model element correlates to a Knowledge Base Item.

Figure 3.7: Meta model of the semantic model



Source: Own illustration

3.4.1 Child benefit from EStG

The claim of child benefit is composed in four different sections of the Einkommenssteuergesetz (EStG) [14]: First in the claim check (§ 62), then in the legal definition of children (§ 63 and §32) and lastly in the amount of child benefit (§ 66).

The child benefit act in the EStG puts focus on arithmetical operations that can hardly be modeled with purely logic-based approaches. The corresponding Section 5.2.1 shows the result of this formalization.

3.4.1.1 Relevant norms

The Listings 3.2, 3.3, 3.4 and 3.5 show the different excerpts from the law. For reasons of simplicity, the law excerpts are not described in detail, but an alternative representation in form of activity diagrams is shown. These diagrams are the precursor for the resulting semantic model with all attributes and constraints. Since the law is composed in German, the activity diagrams as well as the semantic model are also in German.

Listing 3.2: Excerpt from §63 from the EStG

```
(1) Als Kinder werden berücksichtigt
 1. Kinder im Sinne des §32 Absatz 1,
 2. vom Berechtigten in seinen Haushalt aufgenommene Kinder
    seines Ehegatten,
 3. vom Berechtigten in seinen Haushalt aufgenommene Enkel.
[...]
```

Listing 3.3: Excerpt from §32 from the EStG

```
(1) Kinder sind
1.  im ersten Grad mit dem Steuerpflichtigen verwandte Kinder,
2.  Pflegekinder [...]

(2) [...]

(3) Ein Kind [...] [dass] das 18. Lebensjahr noch nicht vollendet
   hat [...].

(4) Ein Kind, das das 18. Lebensjahr vollendet hat, wird
    berücksichtigt, wenn es
1.  noch nicht das 21. Lebensjahr vollendet hat, nicht in einem
   Beschäftigungsverhältnis steht und [...] als Arbeitsuchender
   gemeldet ist oder
2.  noch nicht das 25. Lebensjahr vollendet hat und
     a) für einen Beruf ausgebildet wird oder
     b) sich in einer Übergangszeit von höchstens vier Monaten
        befindet, die zwischen zwei Ausbildungsabschnitten [...]
        liegt, oder
     c) eine Berufsausbildung mangels Ausbildungsplatzes nicht
        beginnen oder fortsetzen kann oder
     d) ein freiwilliges soziales Jahr oder ein freiwilliges
        ökologisches Jahr [...] leistet oder
3.  wegen körperlicher, geistiger oder seelischer Behinderung
    außerstande ist, sich selbst zu unterhalten [...]

[...]
```

Listing 3.4: Excerpt from §62 from the EStG

```
(1) Für Kinder im Sinne des §63 hat Anspruch auf Kindergeld nach
   diesem Gesetz, wer
 1. im Inland einen Wohnsitz oder seinen gewöhnlichen Aufenthalt
    hat oder
 2. ohne Wohnsitz oder gewöhnlichen Aufenthalt im Inland
   a) nach §1 Absatz 2 unbeschränkt einkommensteuerpflichtig ist
      oder
   b) nach §1 Absatz 3 als unbeschränkt einkommensteuerpflichtig
      behandelt wird.
[...]
```

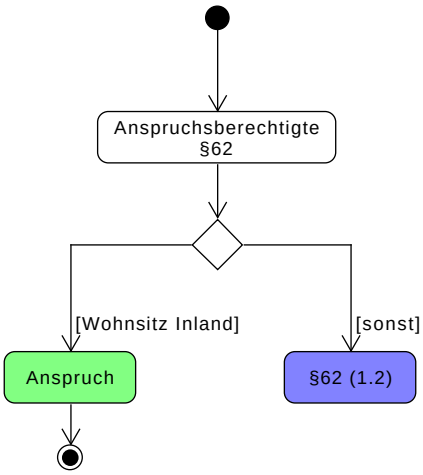
Listing 3.5: Excerpt from §66 from the EStG

```
(1) Das Kindergeld beträgt monatlich für erste und zweite Kinder
   jeweils 190 Euro, für dritte Kinder 196 Euro und für das
   vierte und jedes weitere Kind jeweils 221 Euro.
(2) [...]
```

3.4.1.2 Decision-making structures as activity diagrams

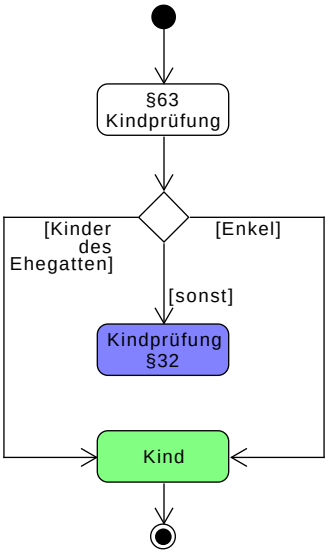
Figures 3.9 and 3.10 show the check, if a child is one in terms of the law and a taxpayer is able to receive child benefit for it. In Figure 3.8 it is checked, whether a taxpayer has a claim in general. The actual calculation of the amount of child benefit is not modeled as activity diagram, but can easily be obtained by the textual description in Listing 3.5.

Figure 3.8: Activity diagram for *Anspruchsprüfung* EStG §62



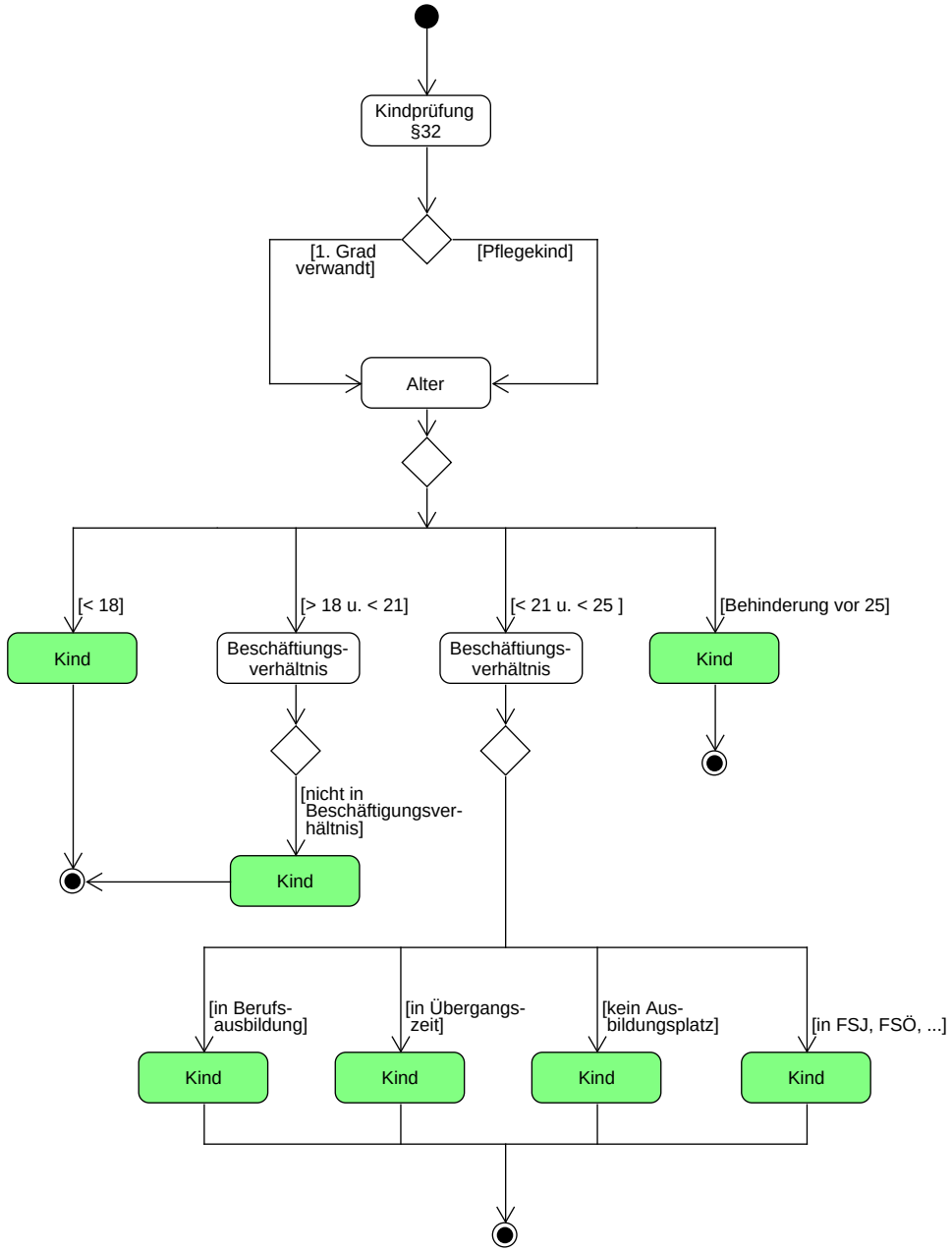
Source: Own illustration, based on the specific law EStG §62

Figure 3.9: Activity diagram for *Kindprüfung* EStG §63



Source: Own illustration, based on the specific law EStG §63

Figure 3.10: Activity diagram for *Kindprüfung* EStG §32



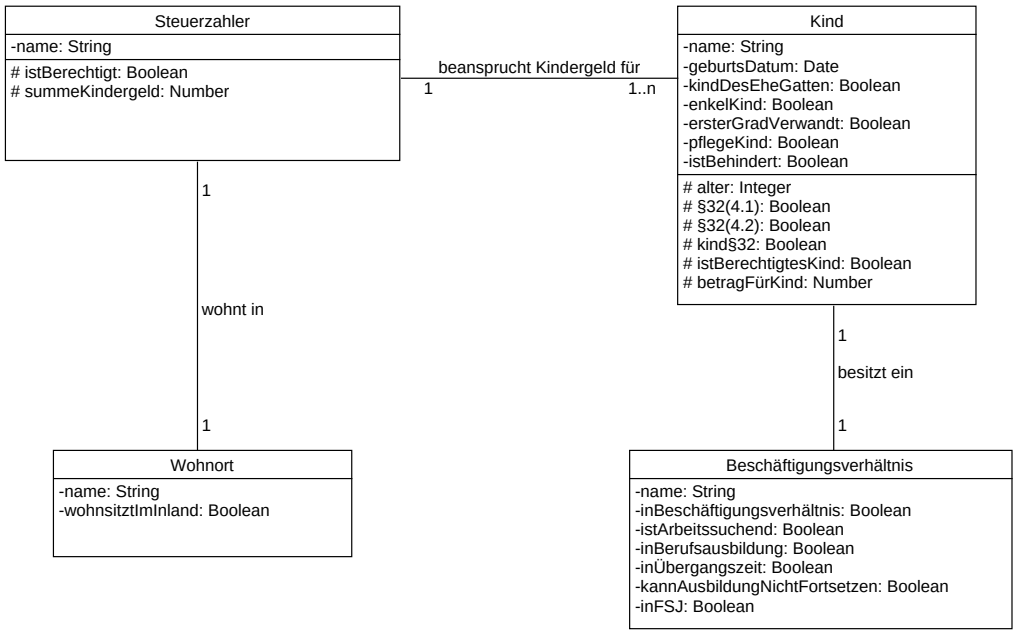
Source: Own illustration, based on the specific law EStG §32

3.4.1.3 Semantic model

Figure 3.11 shows the semantic model, which should be the result of the formalization process. This model shows the structure of the norm as well as the relationships between the entities. Furthermore, the attributes and their data types are displayed. The attributes marked with # are derived attributes (defined as MxL expressions).

The types *Steuerzahler* (taxpayer), *Kind* (child), *Wohnort* (residence) and *Beschäftigungsverhältnis* (employment status) have been identified. The model is from the taxpayer's point of view. The taxpayer can have an association to one or more children with each of them having an individual employment status. Furthermore, the taxpayer has a reference to a residence where he or she lives.

Figure 3.11: Target model for child benefit claim



Source: Own illustration

3.4.1.4 Mathematical definition

This section describes the rules and conditions that must be translated for the target model in MxL expressions.

Type definitions

$$s \in Steuerzahler \tag{3.1}$$

$$k_j \in Kind,\ j \in \mathbb{N} \tag{3.2}$$

$$w \in Wohnort \tag{3.3}$$

$$b \in Beschäftigungsverhältnis \tag{3.4}$$

Relations

$$\begin{aligned} &beanspruchtKindergeld \subseteq Steuerzahler \times Kind : \\ &\quad (s, k_j) \in beanspruchtKindergeld \\ &\quad \implies k_j \text{ ist das } j. \text{ Kind von Steuerzahler } s \end{aligned} \tag{3.5}$$

$$\begin{aligned} &wohntIn \subseteq Steuerzahler \times Wohnort : \\ &\quad (s, w) \in wohntIn \implies Steuerzahler\ s \text{ wohnt in Ort } w \end{aligned} \tag{3.6}$$

$$\begin{aligned} &besitztEin \subseteq Kind \times Beschäftigungsverhältnis : \\ &\quad (k_j, b) \in besitztEin \implies Kind\ k_j \text{ besitzt das } Beschäftigungsverhältnis\ b \end{aligned} \tag{3.7}$$

Rules

$$k_j.alter := \lfloor NOW - k_j.geburtsDatum \rfloor \tag{3.8}$$

$$\begin{aligned} k_j.istBerechtigtesKind := (&k_j.kindDesEheGatten \\ &\vee k_j.enkelKind \vee k_j.ersterGradVerwandt \\ &\vee k_j.pflegeKind) \wedge k_j.kind\S 32 \end{aligned} \tag{3.9}$$

$$k_j.kind\S32 := k_j.\S32(4.1) \vee k_j.\S32(4.2) \vee k_j.istBehindert \vee k_j.alter < 18 \tag{3.10}$$

$$\begin{aligned} k_j.\S32(4.1) := {} & k_j.alter > 18 \wedge k_j.alter < 21 \wedge \\ & \neg b.inBeschäftigungsverhältnis \wedge \neg b.istArbeitssuchend \end{aligned} \tag{3.11}$$

$$\begin{aligned} k_j.\S32(4.2) := {} & k_j.alter > 18 \wedge k_j.alter < 25 \\ & \wedge (b.inBerufsausbildung \vee b.in\ddot{U}bergangszeit \\ & \vee b.kannAusbildungNichtFortsetzen \vee b.inFSJ) \end{aligned} \tag{3.12}$$

$$\begin{aligned} K^s := \{ & k_j \in Kind | (s, k_j) \in beanspruchtKindergeld \\ & \wedge\ k_j.istBerechtigtesKind\} \ für\ s \in Steuerzahler \end{aligned} \tag{3.13}$$

$$\begin{aligned} s.istBerechtigt := {} & w.wohnsitzImInland\ für\ (s, w) \in wohntIn, \\ & s \in Steuerzahler,\ w \in Wohnort \end{aligned} \tag{3.14}$$

$$s.summeKindergeld := \begin{cases} \sum_{k_j \in K^s} k_j.betragFürKind & falls\ s.istBerechtigt \\ 0 & falls\ \neg s.istBerechtigt \end{cases} \tag{3.15}$$

$$k_j.betragFürKind := \begin{cases} 190 & falls\ j = 1\ oder\ j = 2 \\ 196 & falls\ j = 3\ oder\ j = 4 \\ 221 & falls\ j > 4 \end{cases} \tag{3.16}$$

3.4.2 Reporting obligation from BDSG

The second case study is the reporting obligation according to §4 of the Bundesdatenschutzgesetz (BDSG). This act controls whether the processing of personal data must be reported to authorities or not [13]. Besides the textual representation, the activity diagrams in Figure 3.12 and Figure 3.13 depict the necessary actions and decisions. This regulation was chosen, because it exemplary contains a lot of logic-based decisions which must be evaluated correctly.

3.4.2.1 Relevant norms

The main norm for the regulation is the §4 in the BDSG, which is presented in Listing 3.6.

Listing 3.6: Excerpt from §4d from the BDSG

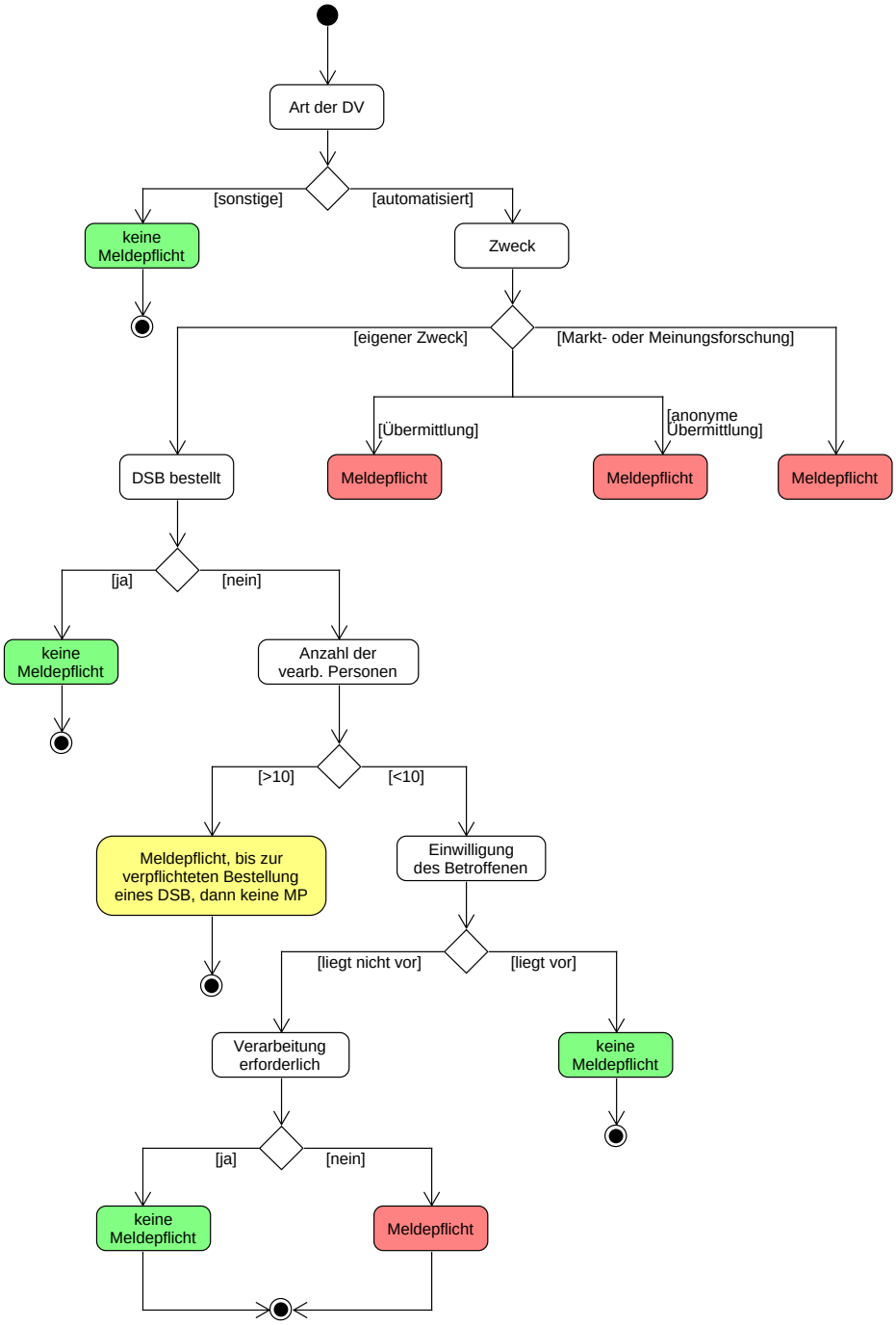
```
(1) Verfahren automatisierter Verarbeitungen sind vor [...] zu
   melden.
(2) Die Meldepflicht entfällt, wenn die verantwortliche Stelle
   einen Beauftragten für den Datenschutz bestellt hat.
(3) Die Meldepflicht entfällt ferner, wenn die verantwortliche
   Stelle personenbezogene Daten für eigene Zwecke erhebt,
   verarbeitet oder nutzt, hierbei in der Regel höchstens neun
   Personen ständig mit der Erhebung, Verarbeitung oder Nutzung
   personenbezogener Daten beschäftigt und entweder eine
   Einwilligung des Betroffenen vorliegt oder die Erhebung,[...]
   erforderlich ist.
(4) Die Absätze 2 und 3 gelten nicht, wenn es sich um
   automatisierte Verarbeitungen handelt, in denen geschäftsmäßig
   personenbezogene Daten von der jeweiligen Stelle
   1. zum Zweck der Übermittlung,
   2. zum Zweck der anonymisierten Übermittlung oder
   3. für Zwecke der Markt- oder Meinungsforschung gespeichert
      werden.
(5) Soweit automatisierte Verarbeitungen besondere Risiken [...]
   aufweisen, unterliegen sie der Prüfung vor Beginn der
   Verarbeitung (Vorabkontrolle). Eine Vorabkontrolle ist
   insbesondere durchzuführen, wenn
   1. besondere Arten personenbezogener Daten (§ 3 Abs. 9)
      verarbeitet werden oder
```

```
   2. die Verarbeitung personenbezogener Daten dazu bestimmt ist,
      die Persönlichkeit des Betroffenen zu bewerten [...],
   es sei denn, dass eine gesetzliche Verpflichtung oder eine
   Einwilligung des Betroffenen vorliegt oder die Erhebung
   [...] erforderlich ist.
(6) Zuständig für die Vorabkontrolle ist der Beauftragte für den
   Datenschutz. Dieser nimmt die Vorabkontrolle nach Empfang der
   Übersicht nach §4g Abs. 2 Satz 1 vor. Er hat sich in
   Zweifelsfällen an die Aufsichtsbehörde [...] zu wenden.
```

3.4.2.2 Decision-making structures as activity diagrams

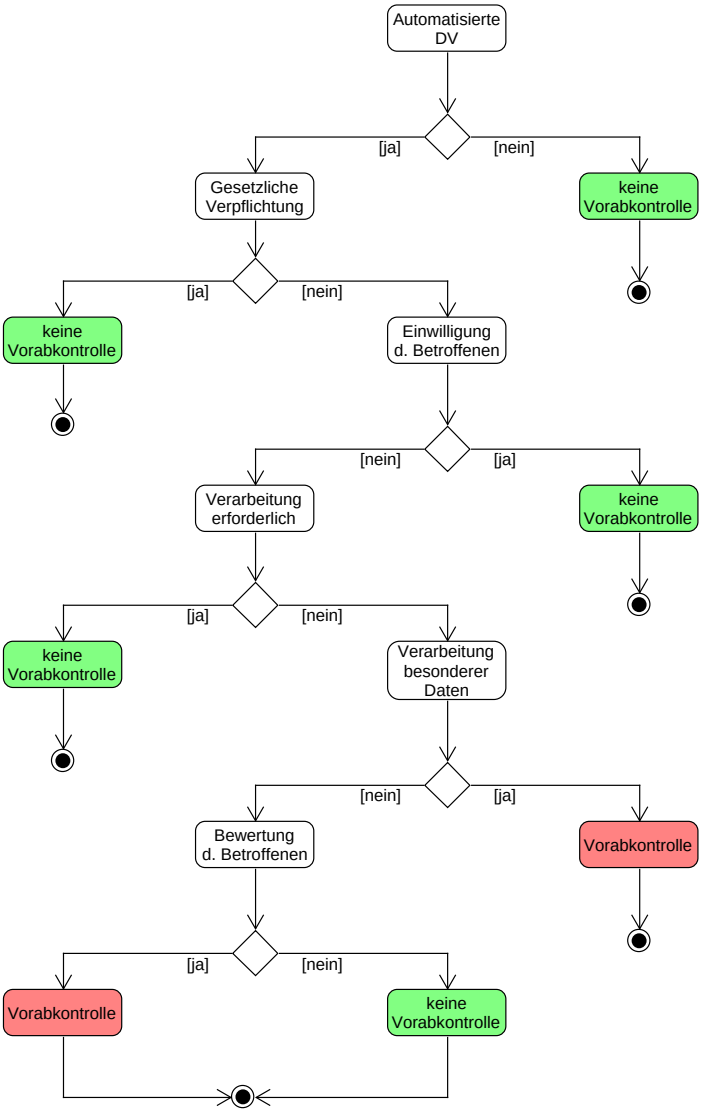
Figure 3.12 shows the activity diagram whether the processing of data must be reported or not (Meldepflicht). The green boxes indicate that there is no reporting obligation, whereas the red boxes point out that a reporting must be done. The yellow box depicts that there is an reporting obligation, but only until a privacy officer has been installed. The regulation about the *Vorabkontrolle* (prior checking) is shown in Figure 3.13.

Figure 3.12: Activity diagram for *Meldepflicht* BDSG §4



Source: Own illustration

Figure 3.13: Activity diagram for *Vorabkontrolle* BDSG §4



Source: Own illustration

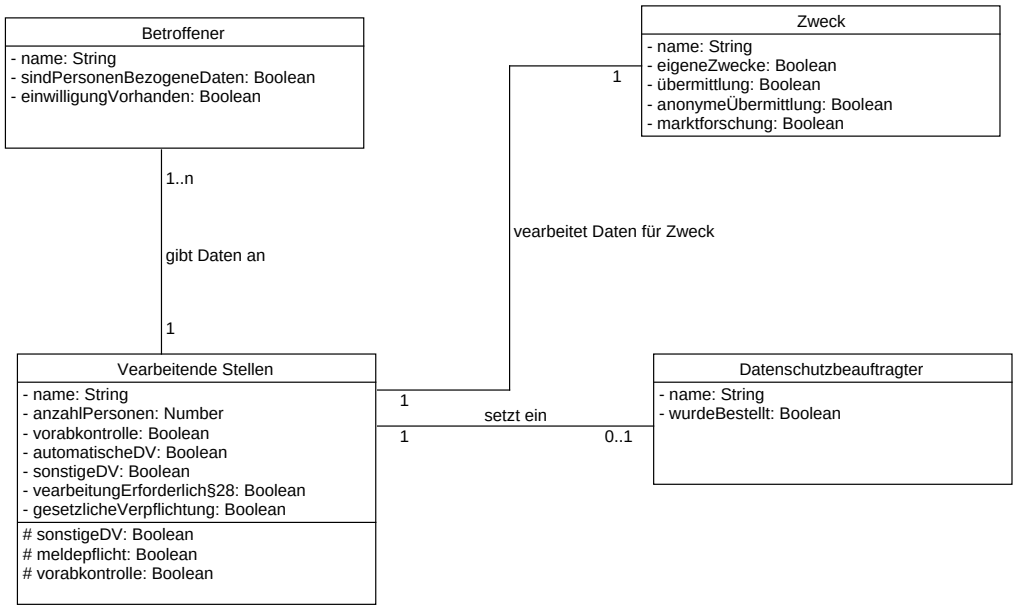
3.4.2.3 Semantic model

Figure 3.14 shows the semantic model for the reporting obligation from the processing authority's point of view. The types in this model are *Betroffener* (affected party), *Zweck* (purpose), *Vearbeitende Stelle* (processing authority) and *Datenschutzbeauftrager* (privacy officer).

The affected party transmits its data to an authority which then processes this data. One authority can receive data from multiple parties. For reasons of simplicity it is assumed that the authority processes all of the data only for one purpose (the purpose is the same for each affected party). Furthermore, the authority can have at most one privacy officer.

For this semantic model the formalization was done in a slightly different way as for the child benefit in Section 3.4.1. Whereas for the first one mainly basic logical equations along with arithmetical expressions for direct usage in the semantic model were defined, this time the rules are composed as implications. These implications should define the conditions, which must be met, such that a reporting obligation results. Hence, for this model only the semantic meaningful derived attributes are defined without any interim results. These implications must later then be transformed into a set of logical expressions, that are reasonable by MxL and SocioCortex.

Figure 3.14: Target model for reporting obligation



Source: Own illustration

3.4.2.4 Mathematical definition

Type definitions

$$b \in Betroffener \tag{3.17}$$

$$v \in VerarbeitendeStelle \tag{3.18}$$

$$d \in Datenschutzbeauftragter \tag{3.19}$$

$$z \in Zweck \tag{3.20}$$

Relations

$$\begin{aligned} &gibtDatenAn \subseteq Betroffener \times VerarbeitendeStelle : \\ &(b, v) \in gibtDatenAn \implies b\ gibt\ Daten\ an\ Stelle\ v \end{aligned} \tag{3.21}$$

$$setztEin \subseteq VerarbeitendeStelle \times Datenschutzbeauftragter : \\ (v, d) \in setztEin \implies v\ setzt\ Datenschutzbeauftragten\ d\ ein \tag{3.22}$$

$$verarbeitet \subseteq VerarbeitendeStelle \times Zweck :\ (v, z) \in verarbeitet \\ \implies Verarbeitende\ Stelle\ v\ verarbeitet\ Daten\ für\ Zweck\ z \tag{3.23}$$

Rules

$$v.sonstigeDV \implies \neg v.meldepflicht \tag{3.24}$$

$$v.automatischeDV \wedge \\ (z.übermittlung \vee z.anonymeÜbermittlung \vee z.marktforschung) \\ \implies v.meldepflicht \tag{3.25}$$

$$v.automatischeDV \wedge z.eigenerZweck \wedge d.wurdeBestellt \\ \implies \neg v.meldepflicht \tag{3.26}$$

$$v.automatischeDV \wedge z.eigenerZweck \wedge \\ \neg d.wurdeBestellt \wedge v.anzahlPersonen >= 10 \\ \implies v.meldepflicht \wedge d.mussBestelltWerden \tag{3.27}$$

$$v.automatischeDV \wedge z.eigenerZweck \wedge \\ \neg d.wurdeBestellt \wedge v.anzahlPersonen < 10 \wedge \\ b.einwilligungVorhanden \implies \neg v.meldepflicht \tag{3.28}$$

$$v.automatischeDV \land z.eigenerZweck \land \neg d.wurdeBestellt \land v.anzahlPersonen < 10 \land v.vearbeitungErforderling \implies \neg v.meldepflicht \tag{3.29}$$

$$v.automatischeDV \land b.besondereArtPersonenbezogenerDaten \land \neg v.gesetzlicheVerpflichtung \land \neg v.vearbeitungErforderlich \land \neg b.einwilligungVorhanden \implies v.vorabkontrolle \tag{3.30}$$

$$v.automatischeDV \land b.bewertungDesBetroffenen \land \neg v.gesetzlicheVerpflichtung \land \neg v.vearbeitungErforderlich \land \neg b.einwilligungVorhanden \implies v.vorabkontrolle \tag{3.31}$$

3.5 Stakeholders

This section gives a short introduction about the different stakeholders using the modeling system in order to help the reader to understand the concerns of them which are addressed in the requirements analysis (see Section 3.6).

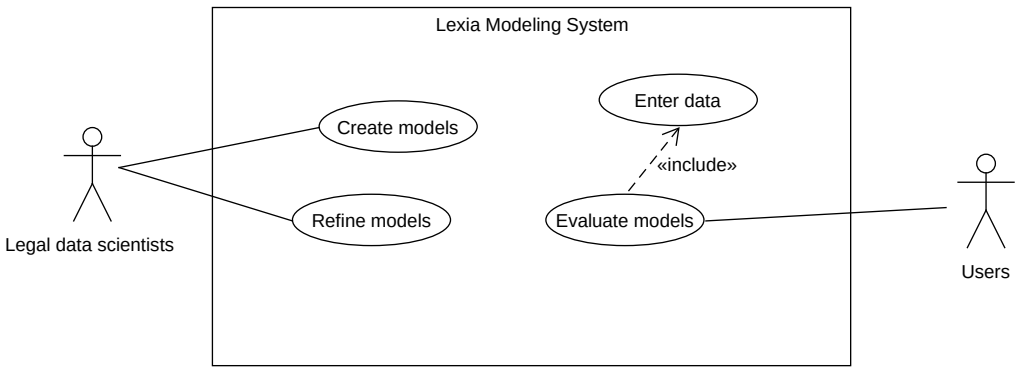
Legal data scientists are users who are responsible for creating and maintaining semantic models. They have a broad knowledge about legal issues and concepts and can formalize their knowledge into semantic models. Their knowledge has been gained through several years of law studies and later through jobs as lawyers and attorneys. Reading, understanding and applying the semantics from normative texts is their daily work. They expect a collaborative tool that supports them during the modeling process.

Users in general do not have the broad knowledge of legal data scientists. In many cases they had an advanced training in law and they are also familiar with basic legal issues. Users are not interested in the legal concepts described in normative texts, but rather in a fast and reliable application of these. A

legal reasoning tool with easy usability as well as an explanation component for decisions are their main expectations.

The concerns of these two different user groups are also roughly sketched in the use case diagram in Figure 3.15. The following Section 3.6 specifies the requirements for the modeling system in detail.

Figure 3.15: Use case diagram depicting different stakeholders



Source: Own illustration

3.6 Requirement analysis

The following requirements were collected during an oral talk [40] with Bernhard Waltl, advisor of this master's thesis, and Corinna Coupette, Research Associate at the Max Planck Institute for Tax Law and Public Finance. Furthermore, the literature research in Section 2.1.2 had resulted in several thoughts which were considered in the requirements analysis.

The requirements analysis is separated in two parts: One consists of requirements to the modeling environment for creating and maintaining semantic models (see Section 3.6.1) and the other part of requirements concerning the evaluation of these models (see Section 3.6.2). At the end, a summary table lists the requirements in a tabular form (see Section 3.6.3). The requirements for the modeling environment are prefixed with *MV-FR*, followed by an increasing number. The same applies for the requirements of the evaluation environment, but *EV-FR* is used as prefix.

As a general constraint, the modeling and evaluation environment must be integrated in Lexia (see Section 3.1.1) which means they should reuse the given components of Lexia (when necessary) and also comply with the current design and layout principles.

3.6.1 Requirements for modeling environment

The following section describes the functional requirements for the modeling environment which is used for creating and maintaining semantic models.

MV-FR 1: Create semantic model management view This management view must operate as entry point of the modeling system. In this view, it must be possible to create semantic models as well as to delete no longer required semantic models. Access to the actual modeling and the evaluation environment must be provided, too.

MV-FR 2: Create view for model workbench The model workbench provides all necessary actions to create a semantic model (see Section 3.3.3). It is the main view, from which a model creator has access to all necessary functions. All following requirements are synthesizing on this view.

MV-FR 3: Support addition and removal of documents In the model workbench, different documents must be addable as well as removable. These documents serve as origin for semantic model elements and they are also needed for providing a linkage between model elements and text.

MV-FR 4: Model workbench must include text view as well as the semantic model During the modeling process the creator must have access to the source text and to the actual semantic model. Ideally there should be no switch of the view between accessing the text and the model. It is favored to use a split screen layout which shows the text and the semantic model simultaneously.

MV-FR 5: Graphical model editor must support pan and zoom For a convenient modeling process, the drawing area for the semantic model must support pan and zoom. This should promote the user even to create very large semantic models, which are beyond the borders of the drawing area.

MV-FR 6: Provide unified shapes for semantic model elements The shapes of the model elements (types) must always look the same in order to achieve a unified look and feel across different semantic models. This should help untrained users to orient themselves even if they are working with unfamiliar semantic models.

MV-FR 7: A semantic model can have any number of types A semantic model can have any number of different type elements. Each type must have an unique name. A semantic model without any types is also valid and can be saved. Ideally a warning message appears before saving, if the model has no types. Previously created types can also be deleted.

MV-FR 8: Types can be linked with imported source text in a many-to-many relationship Any type of a semantic model can be linked with multiple sources from the text, if the source text has been added to the model (see requirement *MV-FR 3*). As described in Section 2.1.2, this concept is necessary due to reasons of validation and traceability. Previously created links between model elements and text must also be revocable.

MV-FR 9: Link must either refer to whole document, to a section of a document or to an annotation Each link between model element and text must either refer to a complete document, to a section within a document (e.g. in case of a law a link to an article) or to a custom annotation. The annotations must be created previously.

MV-FR 10: User must be able to highlight text linked with model elements The user must have the possibility to expose the link of an existing model element. The text, which is linked with the model element, must then

be highlighted in the document. This should help to create the same comprehension of the legal concepts for multiple users working on the same semantic model.

MV-FR 11: A type can have any number of attributes A type can have any number of attributes, including zero attributes. Each attribute is defined by a type-wide unique name and a valid data type. Valid data types are *String*, *Boolean*, *Date*, *Number*, *Longtext* and *MxL-Expression*. In case of a MxL-Expression, a code editor must be displayed, which must support syntax highlighting of these expressions. For the latter feature, the SocioCortex research group provides a component on their GitHub account which should be used[12]. Each attribute can have any number of links to the text, such as described for types in *MV-FR 8*. The constraints for the link target depicted in *MV-FR 9* apply for attributes, too. Attributes can also be deleted.

MV-FR 12: Exactly two different types can be in relation with each other Two model types can be in a bidirectional relation with each other. Each relation must have a source type and a target type and it is also defined by a semantic model-wide unique name. Source and target types must have a valid cardinality. Valid cardinalities are *any*, *at least one*, *at most one* and *exactly one*. Relations can also be linked to the text. For this linkage, the requirements in *MV-FR 8* and *MV-FR 9* apply. A relation can also be revoked, if it is not needed anymore.

MV-FR 13: Summary view for quick outline It should be possible to gain a quick outline over all defined attributes and relations of type, including their linked textual sources. Therefore, a summary view must exist which shows all the necessary information to the user.

MV-FR 14: Semantic model must be savable in Lexia and SocioCortex It must be possible to save the semantic model at any time. The saving must persist the model in Lexia as well as in SocioCortex. While saving, the user

[12] This component is called *mxl-angular* (see https://github.com/sebischair/mxl-angular).

interface should be blocked. After saving is done, the user should be provided with a status notification.

MV-FR 15: Position of types must be changeable The position of types in a semantic model is not fixed. It must be possible to rearrange the model and therefore the position of individual types. This setting should be persisted during saving.

3.6.2 Requirements for evaluation environment

This section depicts the functional requirements for the model evaluation component. This component is used to enter data for a previously created semantic model and perform simple legal reasoning.

EV-FR 1: Three column layout showing semantic model, form and linked text The evaluation view must be structured in a three column layout. The first most left column displays the semantic model, the middle one a form to enter data and the right one contains references to the linked text. The model should be non interactive, meaning that it is fixed and cannot be altered. The types in the model element can be selected which results in a visual highlighting of them.

EV-FR 2: Form must be rendered based on type selection in semantic model The form for entering the input data must be rendered based on the current selection of the type element (see requirement *EV-FR 1*). For each attribute (except for attributes of data type *MxL-Expression*) of a type, a individual input field must appear, labeled with the name of the attribute. Restricting the form just on one specific type helps the user to keep focus on current scope and avoid confusion by large input forms. An icon at the type in the semantic model should indicate, if a user has entered data for this type.

EV-FR 3: Provide mechanism for entering multiple data sets for a type For entering multiple instances of a type (e.g. for the calculation of child benefit if one has more than one child), the form should provide a mechanism for that behavior. Individual data sets must be removable and previous entered

data should also be editable afterwards. A switching mechanism between the single instances must be provided, too.

EV-FR 4: The form input fields must have an input restriction regarding on attribute's data type As described in *MV-FR 11* of the model environment (previous requirements section), different attribute data types are supported. These data types should also be reflected at the input forms. For example, if the user has defined an boolean attribute the input form should only allow to enter truth values or the input field for a number attribute should only allow numerical inputs.

EV-FR 5: Allow references between instances of different types A semantic model can have relations between different types. This must also be reflected on instance level. It must be possible to create links between different instances, but only if their corresponding types have this relation, too. The previously defined cardinality has to be satisfied as well. References must also be editable and revocable.

EV-FR 6: MxL expressions must be evaluated after each action on the instance data The main feature of the evaluation component is legal reasoning. The rules for inference are the MxL expressions, which are defined in attributes. For each attribute with the data type *MxL expression*, a disabled input field is rendered which shows the result of an evaluated expression. The evaluation should occur immediately, if changes regarding type instances (create, update, delete) are performed.

EV-FR 7: Linked text for types must be visible After the selection of a type, the linked text sections must be shown in the right column of the three column view. These text sections should have a hyperlink set to the containing documents.

EV-FR 8: Linked text for attributes must be visible Even for attributes it must be possible to see the corresponding text section, linked with one specific attribute. The text sections should only be shown, if the user requests this explicitly.

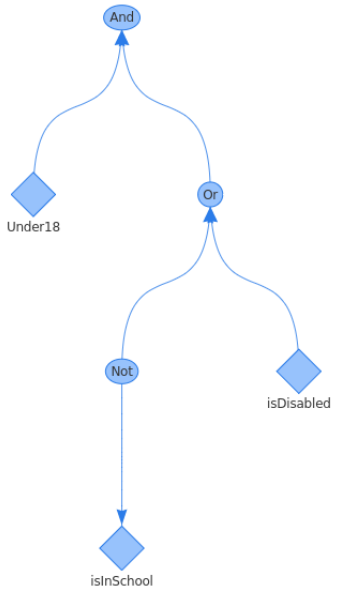
EV-FR 9: Semantic model view must support pan and zoom As described for the modeling environment, the semantic model view in the evaluation environment should support pan and zoom to handle large semantic models, too.

EV-FR 10: Semantic model instances must be savable The data entered in the forms must be saved in SocioCortex, because of the reasoning feature of MxL expressions. Previously entered data must be alterable and deletable.

EV-FR 11: All instance data must be wiped from a semantic model on request It should be possible to clear all model data at once, if a new case should be evaluated and all the old data sets need to be removed.

EV-FR 12: Provide syntax tree for MxL expression In order to understand how a specific decision was made, a syntax tree should visualize the evaluation order of a MxL expression. The analysis of MxL expressions is a built-in feature of SocioCortex and the result of it must only be visualized. An example of such an syntax tree is shown in Figure 3.16.

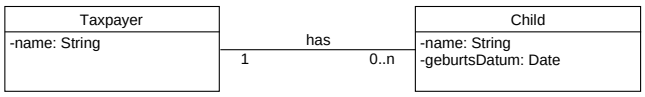
Figure 3.16: Example of a syntax tree



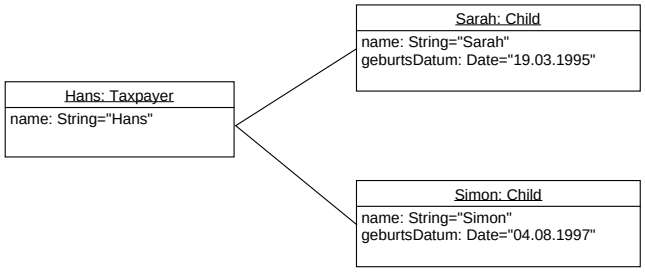
Source: Own illustration

EV-FR 13: Create object diagram view for showing relations between instances An object diagram should help to visualize the relations between different instances. An example of an object diagram with its corresponding data model is shown in Figure 3.17.

Figure 3.17: Example of an object diagram with corresponding data model



(a) An arbitrary data model



(b) Object diagram for data model in (a)

Source: Own illustration

3.6.3 Summary

Table 3.1 summarizes all functional requirements for the modeling view as well as for the evaluation view.

Table 3.1: Summary of all requirements

Number	Requirement
MV-FR 1	Create semantic model management view
MV-FR 2	Create view for model workbench
MV-FR 3	Support addition and removal of documents
MV-FR 4	Model workbench must include text view as well as the semantic model
MV-FR 5	Graphical model editor must support pan and zoom
MV-FR 6	Provide unified shapes for semantic model elements
MV-FR 7	A semantic model can have any number of types
MV-FR 8	Types can be linked with imported source text in a many-to-many relationship
MV-FR 9	Link must either refer to whole document, to a section of a document or to an annotation
MV-FR 10	User must be able to highlight text linked with model elements
MV-FR 11	A type can have any number of attributes
MV-FR 12	Exactly two different types can be in relation with each other
MV-FR 13	Summary view for quick outline
MV-FR 14	Semantic model must be savable in Lexia and SocioCortex
MV-FR 15	Position of types must be changeable
EV-FR 1	Three column layout showing semantic model, form and linked text
EV-FR 2	Form must be rendered based on type selection in semantic model
EV-FR 3	Provide mechanism for entering multiple data sets for a type
EV-FR 4	The form input fields must have an input restriction regarding on attribute's data type
EV-FR 5	Allow references between instances of different types
EV-FR 6	MxL expressions must be evaluated after each action on the instance data
EV-FR 7	Linked text for types must be visible
EV-FR 8	Linked text for attributes must be visible
EV-FR 9	Semantic Model View must support pan and zoom
EV-FR 10	Semantic model instances must be savable
EV-FR 11	All instance data must be wiped from a semantic model on request
EV-FR 12	Provide syntax tree for MxL expression
EV-FR 13	Create object diagram view for showing relations between instances

4 Implementation

This section puts focus on the actual implementation that has been made for the modeling and the evaluation environment. At first, the target system (4.1) is described, followed by an elucidation of the implemented components (4.2), separated in back end (4.2.1) and front end (4.2.2).

4.1 Target system

4.1.1 Architecture

Figure 4.1 depicts the target architecture of the system. The new components, which were implemented, are highlighted in grey (cf. with Figure 3.1 in Section 3.1.1). The additional layer *SocioCortex*, which provides the *Model Storage* and the *MxL reasoning engine*, was connected to Lexia.

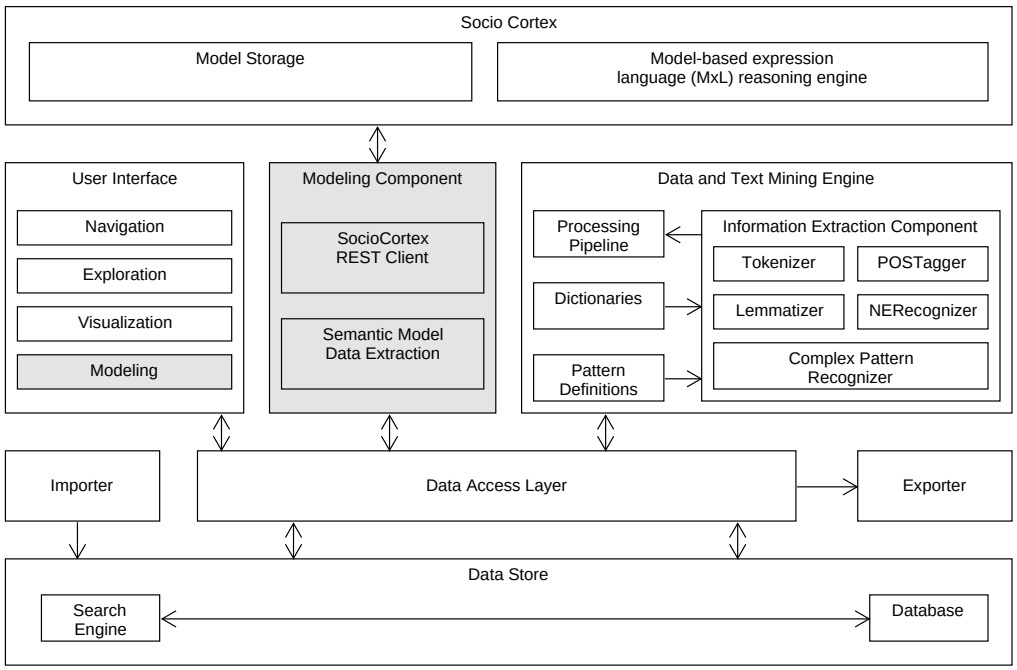
The main contribution to Lexia is the *Modeling Component*, which consists of two other sub components, namely the *SocioCortex REST Client* and the *Semantic Model Data Extraction* component. In addition to that, the user interface was adjusted, too (*Modeling User Interface*).

The *Semantic Model Data Extraction* component is responsible for the extraction of information from the visual representation of a semantic model for further data processing. Its functionality is described in detail in Section 4.2.1.3.

In order to access SocioCortex for persisting the semantic information of the models, a new *SocioCortex REST Client* (see Section 4.2.1.4) was created. This new client simplifies the handling, compared to the existing one in Lexia, since the new REST-client operates with Java entity classes that are serialized to JSON. Before that the JSON objects had to be built manually.

The *Modeling User Interface* is purely implemented in HTML, CSS and JavaScript with *Angular.js*, a framework for creating dynamic single-page applications. *JointJS*[13], a diagramming library, is used as toolkit for drawing the visual representation of the semantic models. Furthermore, *vis.js*[14], a library for generating dynamic browser based visualizations, renders the *object diagram* and the *syntax tree* for MxL expressions.

Figure 4.1: Target architecture of Lexia



Source: Own illustration

4.1.2 Mapping between semantic model elements and SocioCortex entities

One important aspect of the overall work was the transposition of the semantic model from Lexia to SocioCortex, which is the system that evaluates MxL expressions. Hence, a mapping must be created which maps the components of the graphical model representation (types, attributes, relations) to entities of SocioCortex. Table 4.1 shows an approach of how this mapping was implemented.

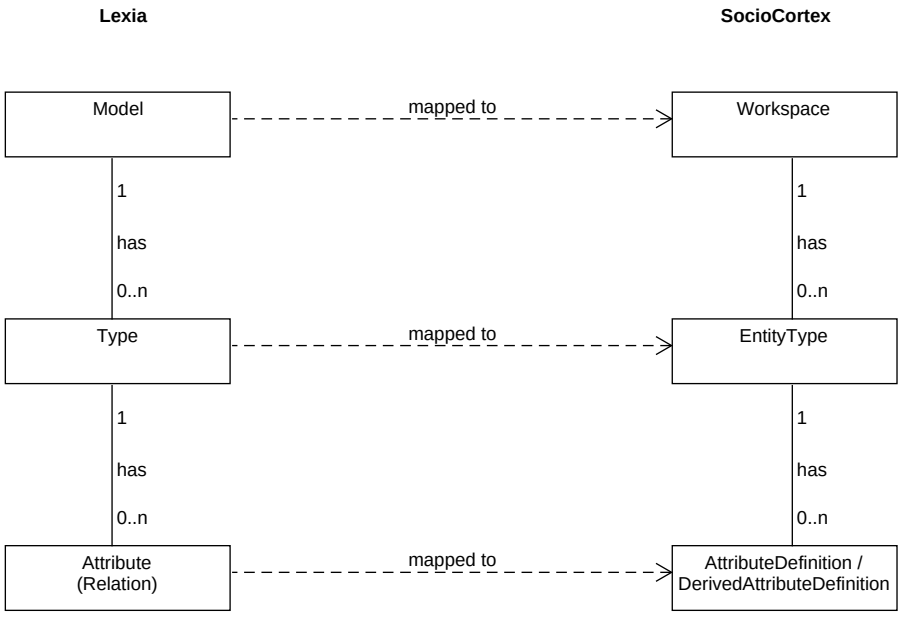
[13] http://www.jointjs.com/
[14] http://visjs.org/

Table 4.1: Mapping between semantic model elements and SocioCortex entities

Semantic model element	**SocioCortex entity**
Model	Workspace
Type	EntityType
Attribute (static) [15]	AttributeDefinition
Attribute (dynamic)[16]	DerivedAttributeDefinition
Relation	AttributeDefinition

This kind of mapping was chosen, because it naturally mirrors the tree structure of the semantic model onto the tree structure of SocioCortex entities. Figure 4.2 compares the hierarchy of the semantic model elements (without relations) in Lexia with their mapped entities from SocioCortex. Depending on their data type (see Table 4.1), *Attributes* are either mapped to *AttributeDefinitions* or to *DerivedAttributeDefinitions*. *Relations* are interpreted as *Attributes* and also mapped to *AttributeDefinitions* in SocioCortex.

Figure 4.2: Tree hierarchy of semantic model elements and SocioCortex entities



Source: Own illustration

[16]Static attributes are all attributes with the data type *String*, *Number*, *Date* or *Longtext* according to requirements in Section 3.6.1.

[16]Dynamic attributes are attributes with data type *MxL* according to requirements in Section 3.6.1.

4.2 Implemented components

The following section describes the changes that were implemented in the practical part of this master's thesis. It is separated in a back end section (see Section 4.2.1) and a front end section (see Section 4.2.2).

4.2.1 Back end

4.2.1.1 Enhancement of data model

The extension of the data model in Lexia for storing the semantic model was easily achievable. Due to the well structured architecture of Lexia extending the base entity class *Entity*, implementing serializer and deserializer for storing the attributes of the class in a map and invoking a *save* function in the super class was sufficient. Listing 4.1 shows the extension of the *Entity* class by *Model*. Concerning the key-value store of the ElasticSearch database, collection attributes (like *List* or *Map*) were limited to a minimum. Otherwise all foreign-key-constrains between the *Model* entity and its collection attributes have to be created and maintained manually. This decision was made after an earlier attempt to normalize the data model and representing *types*, *attributes* and *relations* as own entities turned out to be too complex.

The only collection data type that was used is *relevantDocumentIds*, a list of references to documents, which have been added to a semantic model. This list is converted during serialization into a string, separating the individual IDs with a semicolon.

Another attribute, which has to be saved, is the ID of the corresponding Socio-Cortex workspace (*scWorkspaceId*). This reference is needed for loading the workspace from SocioCortex inclusive all containing sub entities and retrieving the result of evaluated MxL expressions. The *title* attribute represents the title of a model. With two different timestamps (*createdAt* and *updatedAt*) creation and modification time of a model is monitored. At last, the complete serialized semantic model in JSON format is stored in the database. As described earlier, normalizing the data model was too much of an effort and would not lead to any benefits. Moreover, the ElasticSearch database is especially designed for

large unstructured data such as the serialized graph structure in JSON and would therefore not suffer from any performance issues.

Listing 4.1: Excerpt of the *Model* entity

```
1  public class Model extends Entity {
2      private String scWorkspaceId;
3      private String id;
4      private String title;
5      private Date createdAt;
6      private Date updatedAt;
7      private List<String> relevantDocumentIds;
8      private String jsonModelDefinition;
9
10     @Override
11     protected boolean saveEntityElasticsearch() {
12         Map<String, Object> attributes = new HashMap<>();
13
14         if (scWorkspaceId != null && !scWorkspaceId.isEmpty()) {
15             attributes.put("scWorkspaceId", scWorkspaceId);
16         }
17
18         if (!relevantDocumentIds.isEmpty()) {
19             attributes.put("relevantDocumentIds", String.join(";",
                   relevantDocumentIds));
20         }
21
22         ElasticsearchServer.insert(this.SC_TYPE(), attributes);
23     }
24 }
```

4.2.1.2 Format of semantic model

Listing 4.2 shows an overview of the serialized semantic model structure in JSON. This serialization is the result of a built-in function in JointJS[17]. The output is an object with a list of *cells* of different *types*. A cell can either be of type *devs.Model*, which is in the terminology of the semantic model a *type*, or it can be a *link*, which again refers to a *relation*.

The most important attribute of *cells* of type *devs.Model* is *payload*, as it contains all user defined attributes. In the *payload* object, there are not only the

[17] http://www.jointjs.com/api#joint.dia.Graph:toJSON

title of a type and the corresponding identifier of the SocioCortex EntityType (*scId*), but also nested objects for *links*, *attributes* and *instances*.

Besides cells of type *devs.Model* (Line 3-13 and Line 14-24), objects of type *link* (Line 25-42) have a different structure. They contain references to the source and the target element of its relation as well as attributes for the relation such as *multiplicity* of source/target and a *name* for the relation itself.

Listing 4.2: Root level of serialized graph as JSON object

```
{
  "cells": [
    {
      "type": "devs.Model",
      "id": "0c28959a-b1ff-4499-b465-84284743eea2",
      "payload": {
        "links": [],
        "attributes": [],
        "title": "Betroffener",
        "instances": {},
        "scId": "4pnr9jfxdw2j"
      },
    },
    {
      "type": "devs.Model",
      "id": "cf7d1788-47b5-4059-bae4-fe4808f92883",
      "payload": {
        "links": [],
        "attributes": [],
        "title": "Verarbeitende Stelle",
        "instances": {},
        "scId": "138ugwvz74kcp"
      },
    },
    {
      "type": "link",
      "source": {
        "id": "0c28959a-b1ff-4499-b465-84284743eea2"
      },
      "target": {
        "id": "cf7d1788-47b5-4059-bae4-fe4808f92883",
      },
      "id": "4e8d39cc-e9a8-41b0-a486-e3e4eaabbeda",
      "payload": {
        "links": [],
        "multiplicitySource": "*",
        "multiplicityTarget": "1",
        "name": "gibt Daten an",
        "sourceName": "Betroffener",
        "targetName": "Verarbeitende Stelle"
      },
    }
  ]
}
```

An example of a *links* object is shown in Listing 4.3. A link entry consists of a *linkType* (section, annotation or document), a document object (*selectedDocument*) which is referred by the link and depended of the *linkType's* value either an *selectedArticle* or *selectedAnnotation* object. These two objects also contain a text preview of the linked section respectively the linked annotation.

Listing 4.3: *Link* object for a type

```
"payload": {
  "links": [
    {
      "linkType": "section",
      "selectedDocument": {
        "id": "AVbG4L5Thn8z5wU99xrc",
        "title": "Bundesdatenschutzgesetz",
        "titleShort": "Bundesdatenschutzgesetz",
        "documentTypeLowerCase": "law"
      },
      "documentType": "law",
      "selectedArticle": {
        "id": "AVbG4MCjhn8z5wU99xrr",
        "header": "Rechte des Betroffenen",
        "content": "Die Rechte des Betroffenen auf Auskunft [...]"
      }
    },
    {/* next link structure */}
  ],
  "attributes": [ ],
  "title": "Betroffener",
  "instances": {},
  "scId": "4pnr9jfxdw2j"
},
```

The definition of attributes for types is shown in Listing 4.4. An attribute is defined by its *name* and its *datatype*. In case of an MxL attribute, an additional property (*value*) is used for the actual MxL expression. An attribute can also be linked to text. Therefore, it embeds a *link* object (it has the same structure as described for Listing 4.3). The *collapsed* attribute is only for indicating, whether the attributes link view (see Section 4.2.2.2) is expanded or collapsed.

Listing 4.4: *Attribute* object for a type

```
"payload": {
  "links": [],
  "attributes": [
    {
      "name": "personenbezogeneDaten",
      "datatype": "BOOLEAN",
      "collapsed": true,
      "links": []
    },
    {
      "name": "keinePersonenbezogenenDaten",
      "datatype": "MXL",
      "value": "not this.personenbezogeneDaten",
      "collapsed": true,
      "links": []
    }
  ],
  "title": "Betroffener",
  "instances": {},
  "scId": "4pnr9jfxdw2j"
},
```

Lastly, the *instances*, more precisely the actual data instances which were entered by the user, are stored in the graph as well. The benefit of this solution is the negligible effort of joining the instance data with their attribute definitions. Storing the information at one place makes it easier to hold the semantic model and its instantiated data consistent. Furthermore, with the proposed approach of the *visitor pattern* for information extraction (see Section 4.2.1.3), a consistent pattern is used for extracting the semantic model description as well as the populated data from the graph.

The *instances* object shown in Listing 4.5 has a *data* array that has an entry for each individual data set. The actual attribute values for an instance are stored in a flat hierarchy. This flat data structure was used, because of the automatic transformation of JSON to Java classes in the back end. The *ObjectMapper*[18] from *jackson-databind*[19] tries to map all JSON keys to corresponding attributes

[18] https://fasterxml.github.io/jackson-databind/javadoc/2.3.0/com/fasterxml/jackson/databind/ObjectMapper.html

[19] jackson-databind is a library for Java, used for databinding between JSON and Java objects and vice versa. See https://github.com/FasterXML/jackson-databind

of a Java class. This works very well in case of the JSON keys are static, e.g. the *title* attribute is always mapped to the *title* (of data type *String*) in the Java class. However, the names of attributes in the semantic model are dynamic and not fixed. In one case, a user might create an attribute *Herstellungsdatum* and in another case the attribute *AnzahlAngestellte* which has besides different names also different data types. For the transformation of JSON to Java objects the mapper is configured in such a way that unknown attributes, which are not available in the Java class, are automatically added to a map (of data type <String, Object>). After this map has been created, one could use an iterator to iterate over it and extract the attribute names and values. Therefore, the names of the attributes are encoded in the JSON keys. The prefix *SC_ID_* is indicating that the corresponding value is the identifier of an *AttributeValue* in SocioCortex. *SC_NAME* and *SC_ID* are the names and the identifiers of the *Entity* in *SocioCortex*. The *REF_* prefix indicates the value of a relation which contains the identifier and the name of the corresponding counterpart of the relation.

Listing 4.5: *Instance* object for a type

```
"payload": {
  "links": [],
  "attributes": [],
  "title": "Betroffener",
  "instances": {
    "total": 1,
    "current": 1,
    "data": [
      {
        "SC_ID": "l89vvkfu2z9l",
        "SC_NAME": "Dominik",
        "SC_ID_personenbezogeneDaten": "jhdfn5au90cx",
        "personenbezogeneDaten": "true",
        "SC_ID_einwilligungVorhanden": "ptnzjuwywsuh",
        "einwilligungVorhanden": "true",
        "SC_ID_gibt Daten an": "vdepcba9obkq",
        "REF_gibt Daten an": [
          {
            "id": "gog5pz8dzbh4",
            "name": "TUM"
          }
        ]
      }
    ]
  },
  "scId": "4pnr9jfxdw2j"
},
```

Drawback of the overall solution is a poor scalability. Storing the instance data in the semantic model makes it impossible for other users concurrently editing and evaluating this model. If a user changes the structure of a semantic model whilst another user is inserting data, it would not work without any kind of locking mechanism. The implementation should be seen as proof of concept, which can later be enhanced to achieve real concurrent manipulation in a productive environment.

Due to the issue regarding the adaption of the semantic model whilst it contains old instance data (from previous evaluations), the data is completely wiped every time the semantic model is adjusted. This ensures that there are not any remnants of previous evaluations which can conflict with the changed structure

and helps to keep model and instance data consistent. Obviously, the wiping is also performed for the semantic model representation in SocioCortex.

4.2.1.3 Model synchronization between Lexia and SocioCortex

The synchronization between Lexia and SocioCortex is done in a one-way manner, where Lexia is the master system replicating the changes to SocioCortex. For this operation, the *visitor design pattern* is used. Gamma et al. classify the visitor pattern as behavioral pattern and define it as follows: "Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates" [16].

The visual representation of the semantic model is converted from JSON to a Java object in the back end. The structure of this JSON is explained in Section 4.2.1.2. The Java class *JointJsGraph* is the root node containing several other attributes (which are neglected here for the sake of simplicity) including *cells*. A *cell* has all technical attributes for displaying the visual element on the screen (position, size and other attributes) as well as semantic meaningful information (name of element, list of attributes, etc.).

Listing 4.6 shows a shortened version of the container class *JointJsGraph* that can have a list of *cells*. The *cell* class is shown in Listing 4.7. The structure of this class matches exactly the structure of the JSON document shown in Listing 4.2. Both classes also have an *accept* function that is related to the *visitor pattern* and a result of the implementation of the *IModelElement* interface.

Listing 4.6: Excerpt from class *JointJsGraph*

```
public class JointJSGraph implements IModelElement {
    @JsonProperty("cells")
    private List<Cell> cells = new ArrayList<Cell>();

    @Override
    public void accept(IJointJSGraphVisitor visitor) {
        visitor.visit(this);
        for(IModelElement elem : cells) {
            elem.accept(visitor);
        }
    }
}
```

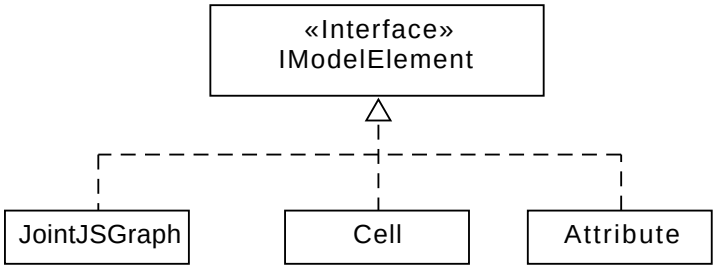
Listing 4.7: Excerpt from class *Cell*

```
public class Cell implements IModelElement {

    @JsonProperty("type")
    private String type;
    @JsonProperty("size")
    private Size size;
    @JsonProperty("position")
    private Position position;
    @JsonProperty("payload")
    private Payload payload;

    @Override
    public void accept(IJointJSGraphVisitor visitor) {
        visitor.visit(this);
        for(IModelElement elem : payload.getAttributes()) {
            elem.accept(visitor);
        }
    }
}
```

With a *visitor*, related operations can be condensed in a class which is then passed to the object structure. When the object structure is traversed and an object has *accepted* the visitor, then the *visit* method is called passing the calling object as an argument. The visitor will then execute the operation specified for this class onto the passed object.

The *IJointJSGraphVisitor* interface, shown in Listing 4.8, has defined three *visit* methods which expect either the complete graph (*JointJSGraph*), a *Cell* or an *Attribute*. The corresponding interface *IModelElement* exports an *accept* function which is implemented in all the above mentioned classes. Figure 4.3 depicts which classes implement the *IModelElement* interface.

Figure 4.3: Classes implementing *IModelElement* interface



Source: Own illustration

Listing 4.8: Interface for *JointJsGraphVisitors*

```
1 public interface IJointJSGraphVisitor {
2     void visit(JointJSGraph graph);
3     void visit(Cell cell);
4     void visit(Attribute attribute);
5 }
```

Listing 4.9: *IModelElement* interface implemented by all model elements

```
1 public interface IModelElement {
2     void accept(IJointJSGraphVisitor visitor);
3 }
```

Figure 4.4 shows the concrete classes that are implementing the *IJointJSGraphVisitor* interface. The tasks, which each visitor performs, are elucidated in the following enumeration:

SocioCortexMetaModelVisitor is responsible for creating meta model elements such as types, attributes and relations in SocioCortex. Therefore, the semantic model object structure is parsed and for each occurrence of a *JointJsGraph* object, a new workspace is created. A *JointJsGraph* only occurs once,

because it is the root node. The same mechanism is used for creating *EntityTypes* and *AttributeDefinitions* in SocioCortex. Relations between model elements, which are transposed into *AttributeDefinitions*, are also created with this visitor.

SocioCortexDerivedAttributesVisitor is used for creating *DerivedAttributes* in SocioCortex. This visitor is invoked after the *SocioCortexMetaModelVisitor*, because the other *AttributeDefinitions* to which is referred to in a *DerivedAttribute* must exist beforehand.

SocioCortexPartialUpdateVisitor extracts instance data from the graph and creates for each instance a corresponding entity in SocioCortex. It is also able to update existing entities by its SocioCortex identifier.

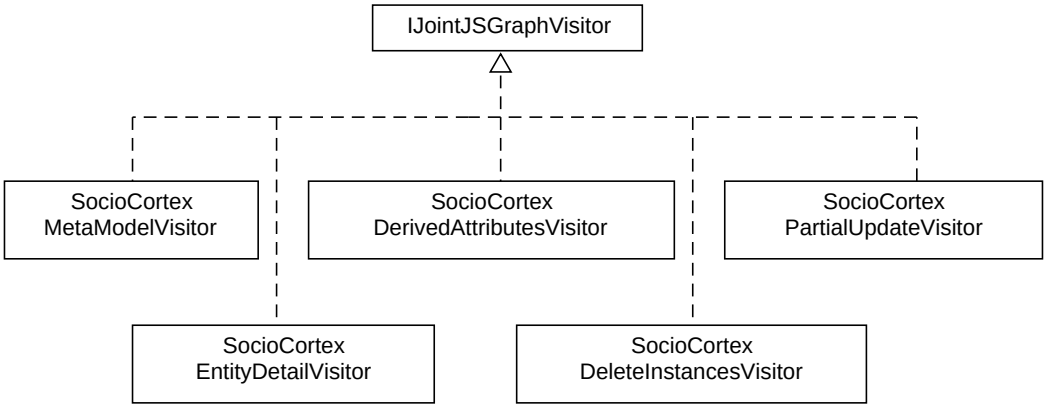
SocioCortexEntityDetailVisitor is used to query information about a workspace from SocioCortex including the evaluated *DerivedAttributes* and merges these information with the semantic model. This merging is done, since the complete instance data is also stored in the graph. In contrast to all other implemented visitors, this one works the other way around, meaning it reads information from SocioCortex and stores them in the graph structure.

SocioCortexDeleteInstancesVisitor deletes all instances from SocioCortex as well as from the graph. This is used for a complete wipe of all data. For deleting only single data instances, a specific REST route was created (see Section 4.2.1.4).

4.2.1.4 Enhancement of REST service

In order to enable the communication between the front end and the back end logic of Lexia, the REST service used for that had to be enhanced. Table 4.2 shows an overview of all routes that were added. In general, there are routes for creating, updating and deleting a semantic model as well as adding and removing data from it. Furthermore, a route for retrieving an evaluated version of a semantic model, in which all derived attributes have been evaluated, was

Figure 4.4: Class hierarchy of visitor implementation



Source: Own illustration

implemented as well. For the syntax tree of MxL expressions a separate route is used which expects as parameter an MxL expression in its HTTP body.

Table 4.2: REST routes for model environment in Lexia

Method	**Route**	**Comment**
GET	/api/model	return all models
GET	/api/model/:id	return a model by *model id*
POST	/api/model	create new model
DELETE	/api/model/:id	delete model by *model id*
PUT	/api/model/:id	update model by *model id*
PUT	/api/model/:id/instance	update model instance by *model id*
POST	/api/model/:id/relevantDocuments	add a document to model by *model id*
DELETE	/api/model/:id/relevantDocuments/:docId	delete a document with *docid* from model by *model id*
DELETE	/api/model/:id/data	delete all populated data from model by *model id*
GET	/api/model/:id/evaluated	fetch the model with evaluated data by *model id*
POST	/api/model/validateMXL	validate an MxL expression
DELETE	/api/model/instance/:instanceId	delete an instance by *SocioCortex' entity id*

4.2.1.5 Model-based REST client for communication with SocioCortex

As proposed in Section 3.3, the actual data is not only stored in Lexia, but also in SocioCortex. Therefore, the REST client in Lexia, communicating with SocioCortex, had to be adjusted. There was already an existing implementation of such a REST client, but all methods for it expected *JSONObjects*, *JSONNodes* and *JSONArrays* as input parameters. This design decision implies that the methods, which are invoking the SocioCortex REST client, must know how the data exchange format must look like. In order to achieve a higher abstraction and a lower coupling, the REST client was changed from this implementation towards a *model-based approach*. This means that the REST client methods accept Java objects as parameters and the actual knowledge about the transformation from this objects to JSON is encapsulated in the REST client (respectively the model classes are constructed in such a way that the serialized JSON corresponds to the same structure SocioCortex expects).

Listing 4.10 shows exemplarily the usage of the SocioCortex REST client in Lexia. For an even more convenient usage, builders were created that construct the Java objects needed as parameters for the REST client.

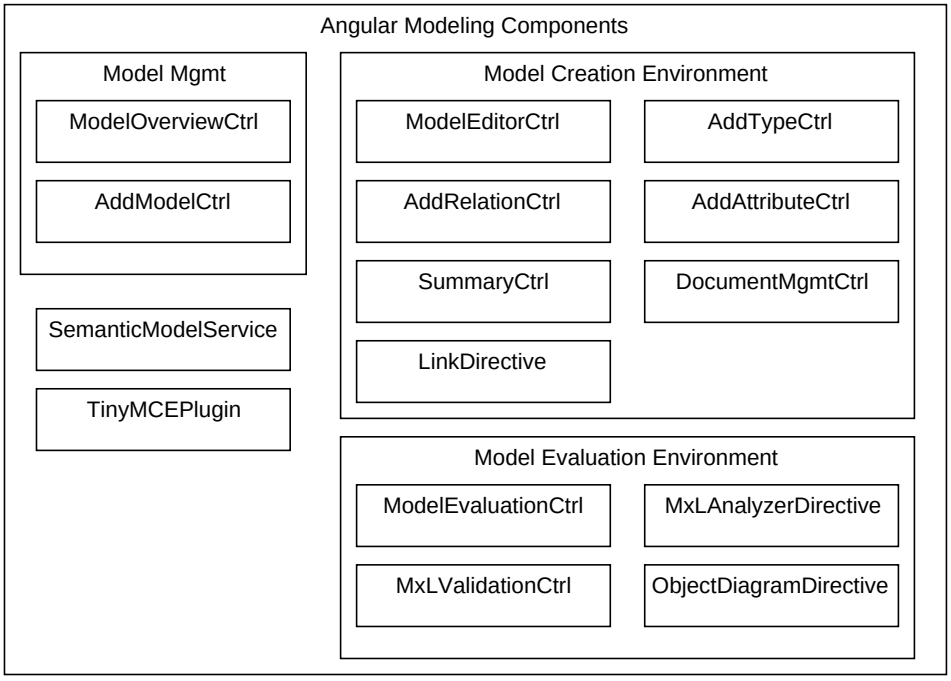
Listing 4.10: Usage of the builder design pattern for the REST client

```
1  // Builder for EntityTypes
2  EntityType entityType = new EntityTypeBuilder()
3                  .setName(title)
4                  .forWorkspaceWithId(currentWorkspace.getId())
5                  .toEntityType();
6
7  SocioCortexControllerV2.createEntityTypeForWorkspace(entityType);
8
9  // Builder for AttributeDefintitions
10 AttributeDefinition attributeDefinition = new AttributeDefinitionBuilder()
11                 .setName(name)
12                 .setEntity(currentEntityType)
13                 .setAttributeType(datatype)
14                 .toAttributeDefinition();
15
16 SocioCortexControllerV2.createAttributeDefinitionForEntityType(attributeDefinition);
```

4.2.2 Front end

Figure 4.5 shows the components of the front end application that were created. They are separated in the four different categories: *Controllers* (and corresponding *views*), *services*, *directives* and *other components*, which are not depending on Angular.js. In the following sections, the individual components are explained.

Figure 4.5: New components of the front end application



Source: Own illustration

4.2.2.1 Angular.js service as wrapper for JointJs

For the usage of the *JointJs JavaScript library* within an Angular.js application, a custom Angular.js service (*SemanticModelService*) was created. This service encapsulates the necessary JointJs functionality and exposes only a well-defined interface. The *SemanticModelService* provides a *createModel* function that accepts as first parameter the id string of a DOM element, on which the model drawing area will be attached to. The second parameter is an object providing callback function handles that are invoked by several events (eg. *onBlankClicked*, *onElementClicked*, *elementMoved*, etc.). With this mechansim,

changes in the graph can be propagated to the corresponding controller that uses the *SemanticModelService*. The last parameter is indicating, whether the model is interactive. In the *ModelEditorCtrl*, this parameter is set to *true*, so that the user can manipulate semantic models, whereas in the *ModelEvaluationCtrl* it is set to *false*.

Another important method is the *addElement* method that creates a new element with a given title and a custom payload. Furthermore, an *onChange* callback and the position where it should initially appear on the drawing area must be provided. Lastly, methods for exporting and importing the JSON structure, needed for serialization and deserialization of the graph, are also provided.

4.2.2.2 Controllers and views

The modeling component provides several Angular.js controllers and views, as well as modal dialogs. In this section, the different views and their functionality are elucidated briefly. Furthermore, screenshots are shown for selected views. Each input field of a form has been equipped with a client-side input validation, so that erroneous inputs are exposed immediately to the user.

ModelOverviewCtrl: The *ModelOverviewCtrl* provides a tabular overview over all semantic models stored in Lexia. This view serves as entry point, from which the user can either navigate to the model editor or to the model evaluation view. Models which are not used anymore can also be deleted.

AddModelCtrl: This view and its associated controller is a modal dialog which is used to create new semantic models. It only shows a text box in which the user must enter the name of the model.

ModelEditorCtrl: The *ModelEditorCtrl* is one of the main components of the system. It enables the creation and refinement of semantic models. Therefore, it has a dependency to the *SemanticModelService* and registers callback functions for different events. The *ModelEditorCtrl* controller can load and save semantic models to the server. The addition or deletion of relevant laws to a semantic model is implemented in this controller as well.

AddTypeCtrl: The *AddTypeCtrl* view is a modal dialog which enables the user to add new types for a given semantic model. Figure 4.7 shows a screenshot of this dialog. It embeds the *Link* directive, which is used to create links between text and semantic model elements (see Section 4.2.2.3).

AddAttributeCtrl: This is also a modal view that opens a dialog in which the user can create and modify attributes for a type. Figure 4.8 shows a screenshot of this view. It uses the *CodeMirror*[20] code editor for syntax highlighting of MxL expressions. Furthermore, the *Link* directive is used in this view for linking attributes to text. Due to reasons of clarity, the *Link* directive as well as the MxL code editor can be collapsed or expanded.

AddRelationCtrl: Modal dialog for creating relations, assigning a name and specifying cardinality of these. Figure 4.9 shows a screenshot of this dialog. The *AddRelation* view also implements the *Link* directive.

SummaryCtrl: After double clicking on a type, the *SummaryCtrl* view is opened and shows a summary of all attributes, relations and links of this type.

DocumentMgmtCtrl: A dialog that enables the addition or deletion of laws for a semantic model. Only the documents that have not been imported yet are shown in this tabular view.

ModelEvaluationCtrl: It is the main view for inserting and deleting data for the semantic model. This view has a dependency to the *SemanticModelService* for rendering the semantic model in a non interactive way. The object diagram for created instances is shown in this view (see Figure 4.10).

MxLValidationCtrl: This popup dialog shows the syntax tree of a given MxL expression. The *MxLValidationCtrl* controller uses the *MxLAnalyzer* directive (see Section 4.2.2.3) to perform its task. A screenshot of the dialog showing the syntax tree is depicted in Figure 4.11.

[20] https://codemirror.net/

Figure 4.6: Screenshot of modeling environment with a highlighted text reference of a type

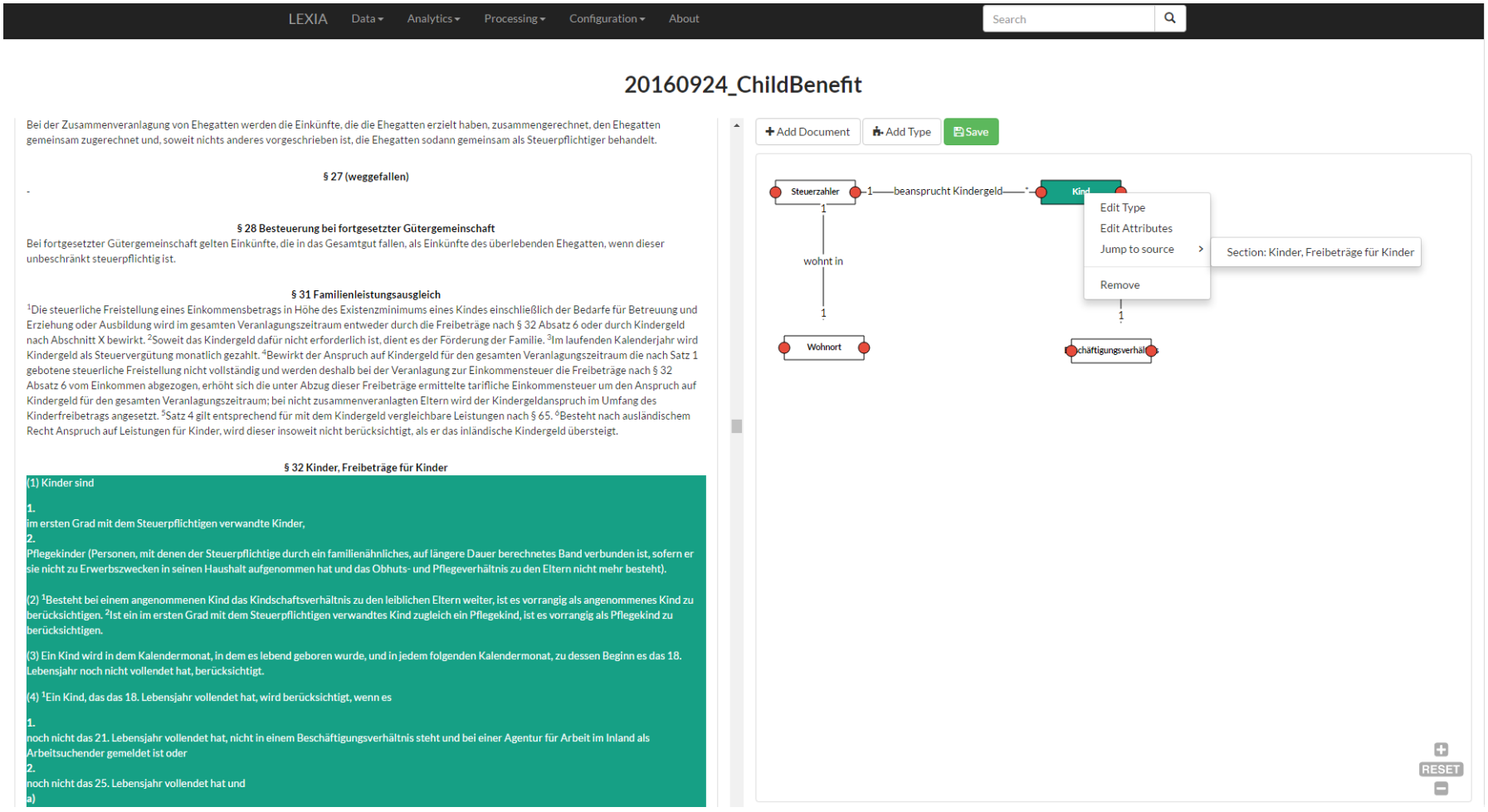


Figure 4.7: Screenshot of the definition of a type

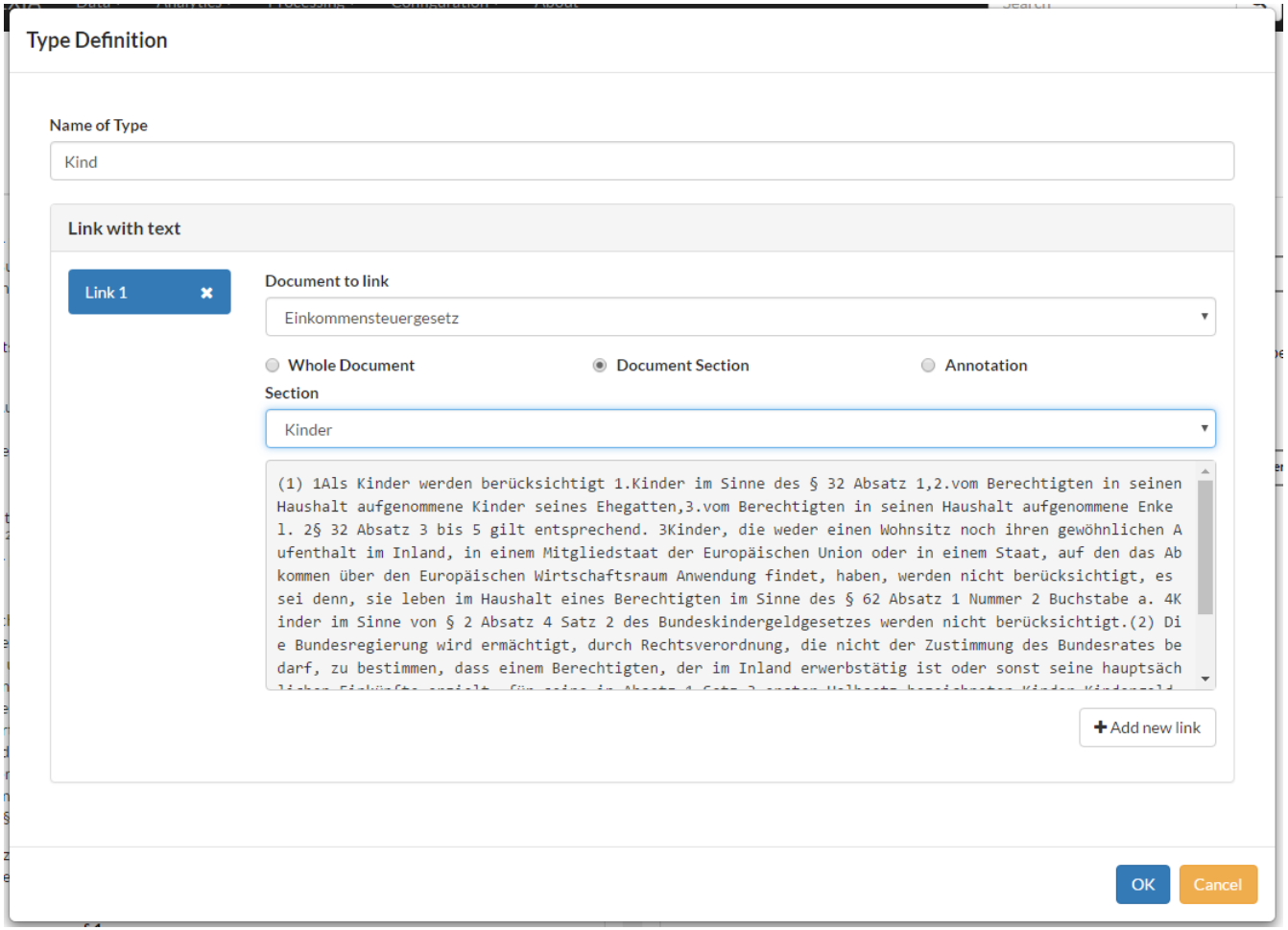


Figure 4.8: Screenshot of the definition of an attribute

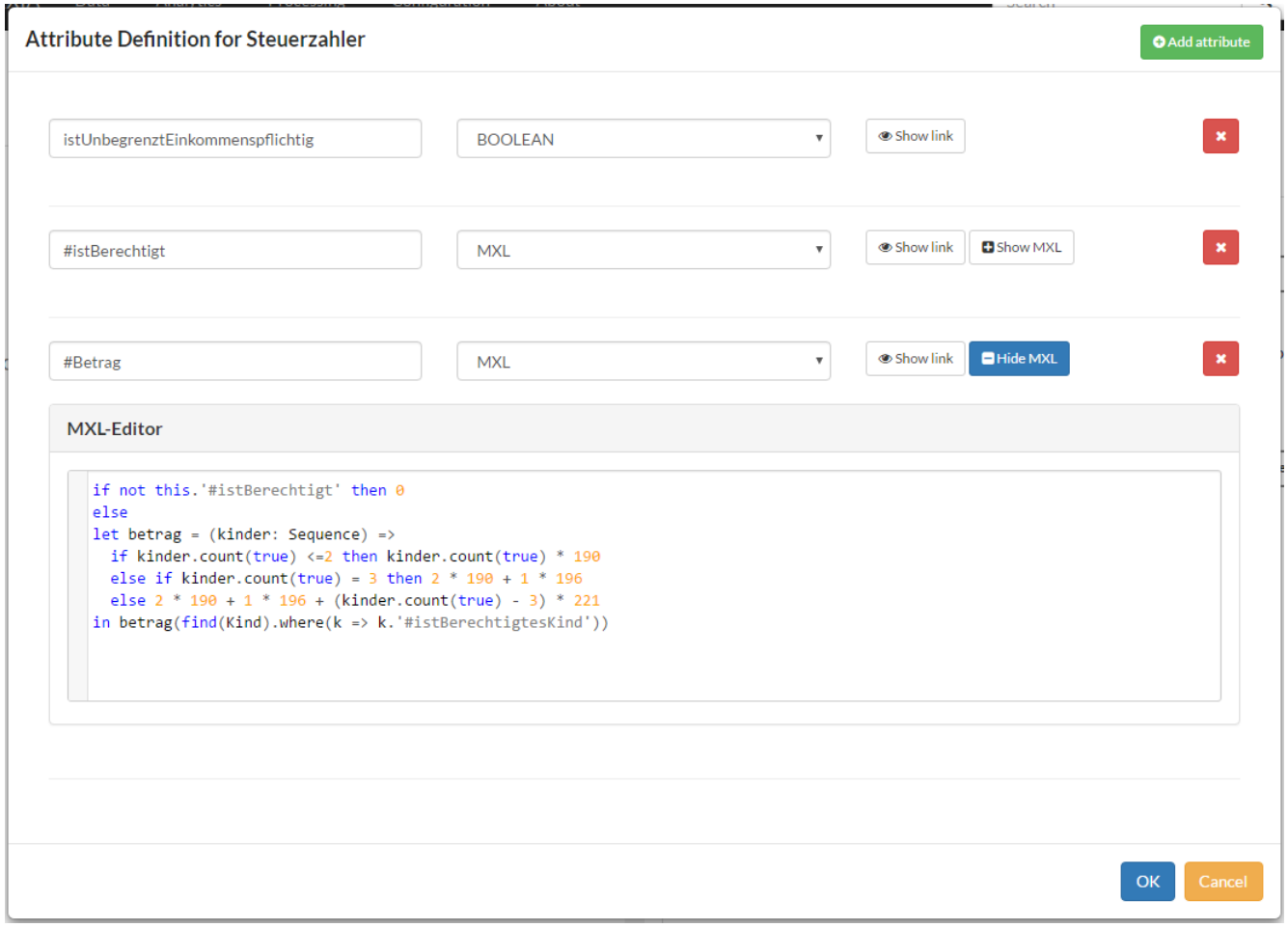


Figure 4.9: Screenshot of the definition of a relation

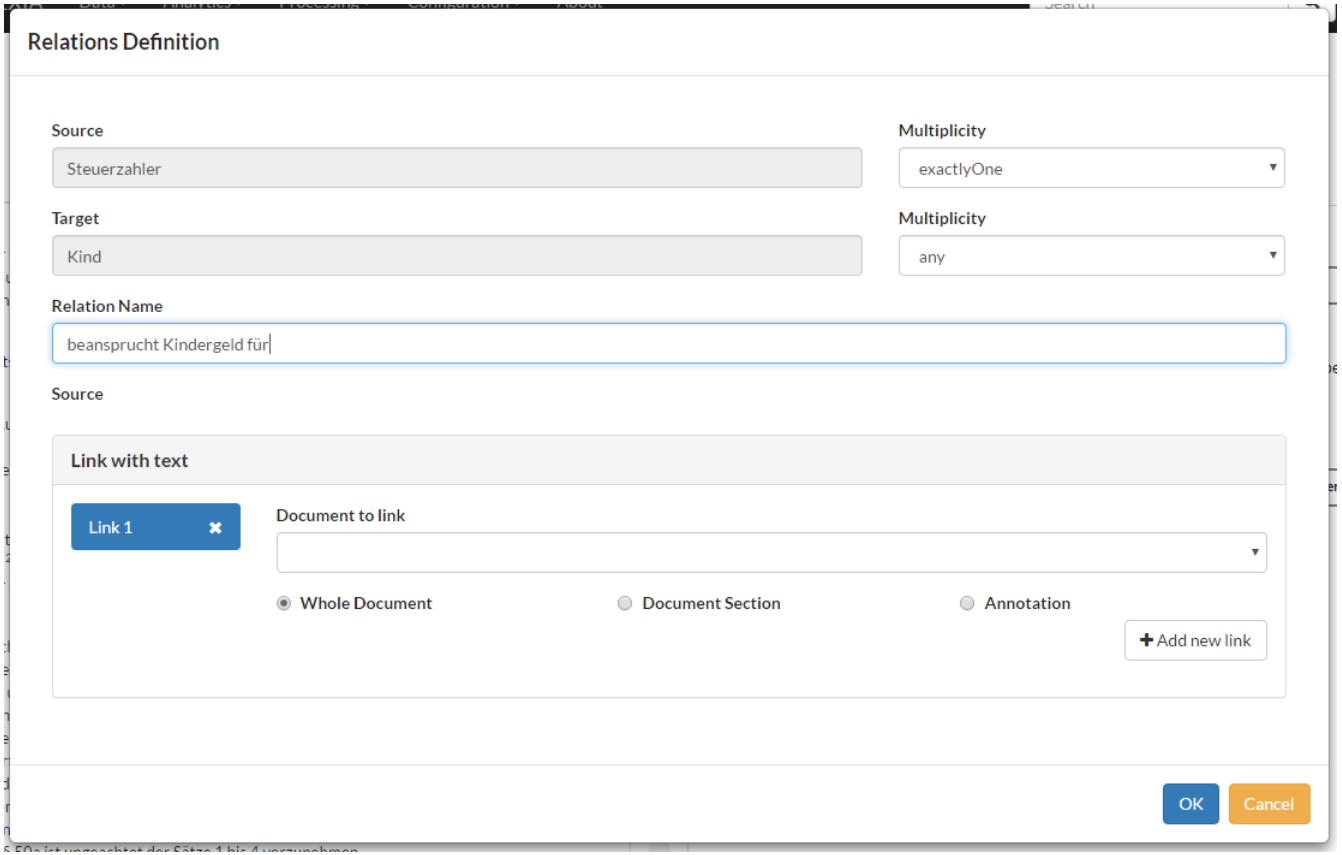


Figure 4.10: Screenshot of the model evaluation environment with a selected type and populated data

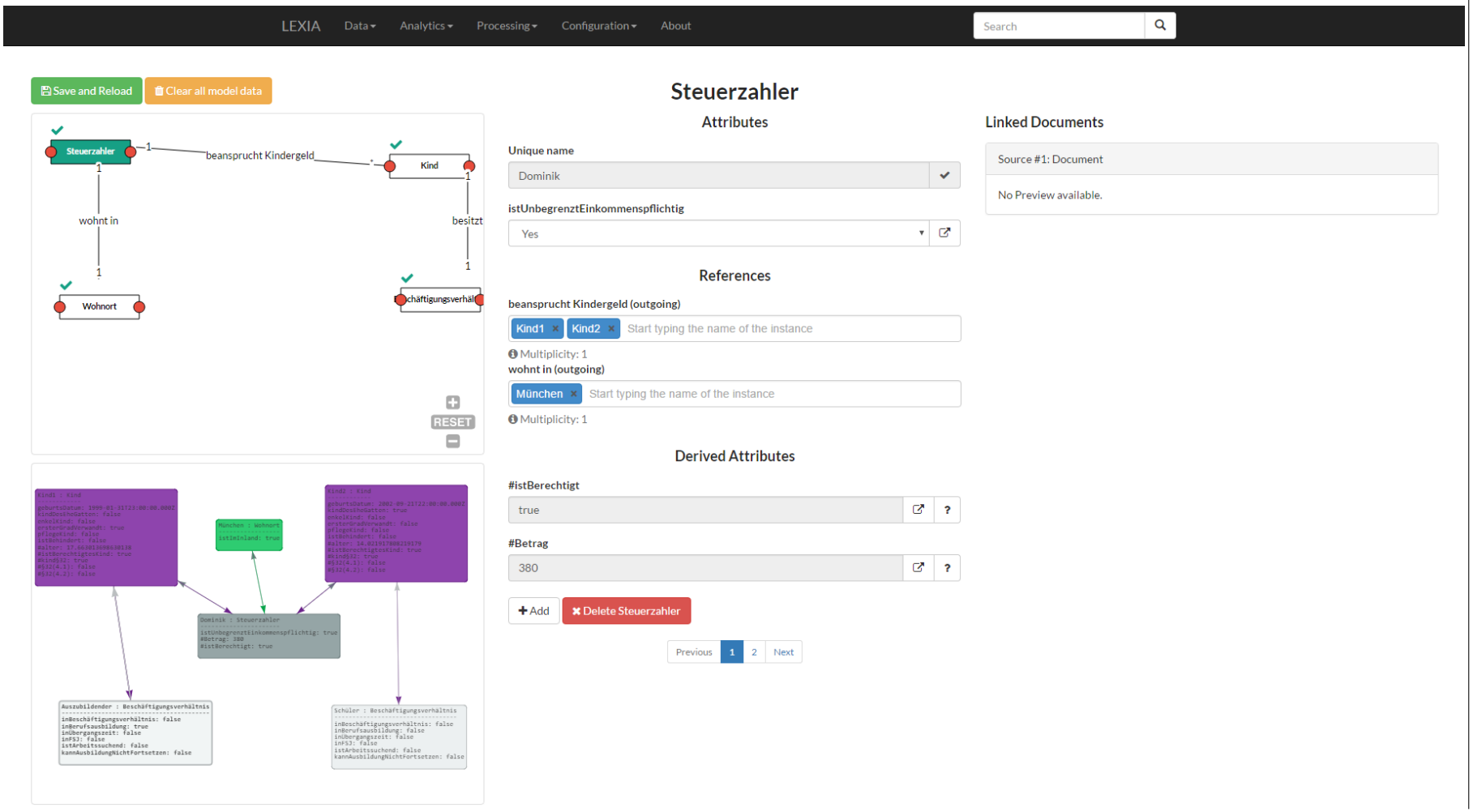
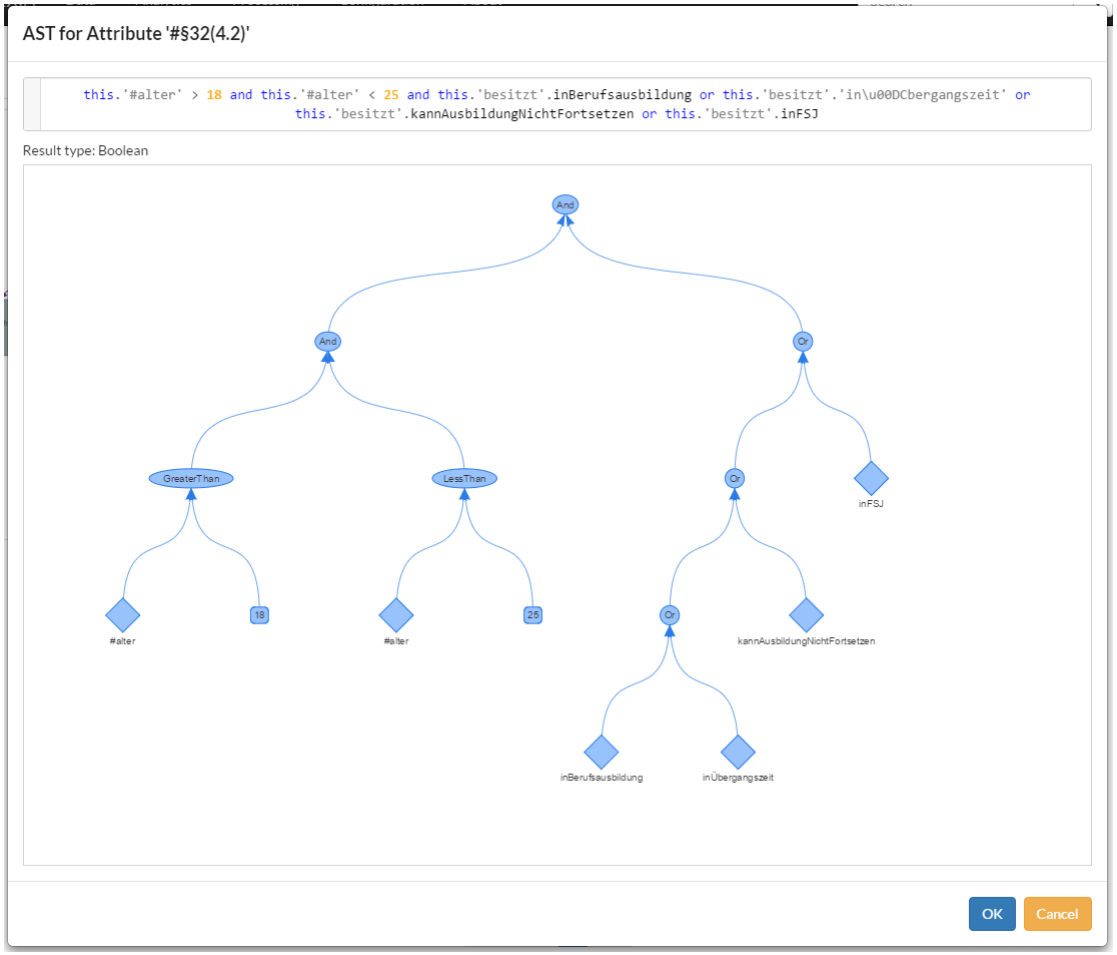


Figure 4.11: Example of an syntax tree for a MxL expression



4.2.2.3 Directives

Link: In order to create a link between semantic model elements and text, a separate directive was created. This directive can be reused in every view where the link feature must be implemented (e.g. creation of types, attributes, relations). Listing 4.11 shows the usage of this directive. The *documents* attribute refers to a list of documents that are JSON objects representing *LegalDocuments* (see Section 3.1.1). The list of documents is used for extracting all sections and annotations out of it. The attribute *links* is the exported link object which contains the selected document, section or annotation which is the target of the link. The structure of this JSON object is shown in Listing 4.3.

Listing 4.11: Usage of the link directive

```
1 <link-to-document documents="documents"
      links="element.links"></link-to-document>
```

The lower section of Figure 4.7 (the box with the title *"Link with text"*) shows the rendered directive. It provides a drop down list for all imported documents as well as selection for the link target (whole document, document section or annotation). For a document section or an annotation a preview is given. With the button on the bottom right of the box (*"Add new link"*), multiple links can be created for an element. The blue vertical tab bar on the left indicates the number of links and also provides the possibility to switch between and delete them.

MxLAnalyzer: The *MxLAnalyzer* is an Angular.js directive which generates a syntax tree out of a MxL validation result. This result can be obtained by requesting the proper REST resource from SocioCortex with a given MxL expression. This directive has a dependency to *vis.js*[21], which is a browser based visualization library for rendering the syntax tree. Listing 4.12 shows the usage of this directive.

Listing 4.12: Usage of the mxl-analyzer directive

```
1 <mxl-analyzer mxl-validation="mxlValidationExpression"></mxl-analyzer>
```

[21] http://visjs.org/

ObjectDiagram: The *ObjectDiagram* directive renders an object diagram for an instantiated semantic model. The Listing 4.13 shows the usage of the directive. It expects two parameters: A semantic model definition in JSON and a function handle which will be invoked after an element of the object diagram has been selected. This is used for synchronization between object diagram and form in order to switch to the selected instance.

Listing 4.13: Usage of the object-diagram directive

```
1 <object-diagram model="currentModel.jsonModelDefinition"
2                 element-selected="onObjectDiagramInstanceClicked">
3 </object-diagram>
```

4.2.2.4 Other components

TinyMCEPlugin: At the time of writing this thesis another student has been working on a contract drafting environment for Lexia. This implementation uses the *TinyMCE* rich text editor [22] for providing enhanced input fields to the user. For this editor a plugin was created that allows linking a semantic model to a contract. It has the purpose of referring to executable semantics described in the contract and also formalized as semantic model. The reader of this contract can directly use the linked semantic model.

[22] https://www.tinymce.com/

5 Evaluation

In this evaluation chapter the research questions, proposed at the beginning of this thesis, are answered (see Section 5.1). Afterwards the focus is set on an evaluation of the case studies (see Section 5.2) which were introduced in Section 3.4. With these case studies it should be proven that the implemented modeling system of Lexia is able to formalize the semantics defined for these two examples. Finally, the limitations of the modeling system are shown which were revealed during the implementation phase and the usage of the system (see Section 5.3).

5.1 Research questions

What are the possibilities of formalizing semantics of normative texts?

At the beginning of Section 2.2, different models for knowledge representation were introduced. Popular approaches are for example formalizations based on rules with different abstraction levels and different expressiveness, logic-based approaches and ontologies in combination with description logics. As argued in Section 3.3, in this thesis a *semantic model* is used that shows entities with attributes and their relationship to each other. This approach is influenced by ontologies, which also represent the knowledge as concept terms with relations and attributes, but instead of description logics a rule-based expression language is used. This expression language supports arithmetical operations which are hardly expressible with description logics.

Which components are required to enhance Lexia for fulfilling the task of model formalization and evaluation?

At the previous status quo, Lexia needed enhancement in the back end as well as in the front end for fulfilling the task of creating and evaluating semantic

models. Section 4.2 describes the development of these components in detail. In summary, a graphical model editor is needed which allows the creation of semantic models in a unified way. For storing these visual representations of the semantic models, the REST routes of the Lexia back end had to be adjusted. Hence, saving, updating, deleting as well as requesting evaluated models from the back end are possible.

The main effort had to be spent at the information extraction component for the graphical model. Thus, the information about types, attributes and relations could be used for further processing such as the creation of corresponding SocioCortex entities. Regarding the communication with SocioCortex, an improved REST client was created that uses Java entity classes as templates for JSON serialization.

How could an approach look like to link elements of a formal model with its textual representations?

This thesis uses a pragmatic approach to link elements of a formal model with text. Due to the hierarchical structure of legal documents and the consistent storage with the same structure in Lexia, every textual unit has its own Java entity. Section 3.1.1 describes this hierarchical document storage with *LegalDocument*, *Section* and *Annotation* as the smallest textual unit that can be arbitrarily created.

A semantic model element can be linked to any number of previously created textual units using the (database) ID as foreign key. The integrity of these foreign-key-relationships has to be maintained manually, since the database system of Lexia (ElasticSearch) is a key-value-store and does not support foreign-key-relations.

With this approach it is possible to link an element to any number of arbitrary textual sources. Drawbacks are the missing foreign key constraints between the model entities and the textual units.

How would a concrete formalization of two selected case studies (child benefit of EStG and reporting obligation of BDSG) in the newly implemented modeling environment of Lexia look like?

This question is answered in detail in Section 5.2 where the results of the individual case studies are presented.

What would be a suitable meta model for the representation of formalized semantics in SocioCortex?

SocioCortex is a hybrid wiki with a dynamic meta model. To represent the semantic model described in this thesis, its meta model had to be mapped to SocioCortex entities. This mapping is described in detail in Section 4.1.2 and shortly summarized in the following paragraph.

The meta model of the semantic model contains the elements *type*, *attribute* and *relation*. As a container embedding these elements, *model* is used. The mapping enforces a transposition of this *model* container to a *Workspace* in SocioCortex. *Types* are mapped to *EntityTypes* and *relations* to *AttributeDefinitions*. A differentiation is made for *attributes*: Static attributes with data type *String*, *Boolean*, *Number*, *Date* or *Longtext* are also transposed to *AttributeDefinition*, whereas dynamic attributes of data type *MxL* are transposed to *DerivedAttributeDefintions*.

What are the benefits of separating the task of creating a formal model and applying it?

Through the separation of the task of model creation and model evaluation, different stakeholders are able to work with the system. On the one hand there are the legal data scientists with their broad knowledge who can work in a collaborative way and create semantic models from normative texts. On the other hand these models with their unified appearance and support for legal reasoning can be used by unexperienced users, whose main focus is applying the law. An administration secretary in a civil service for instance can process applications based on several norms and perform legal reasoning with a semantic model previously created by legal experts.

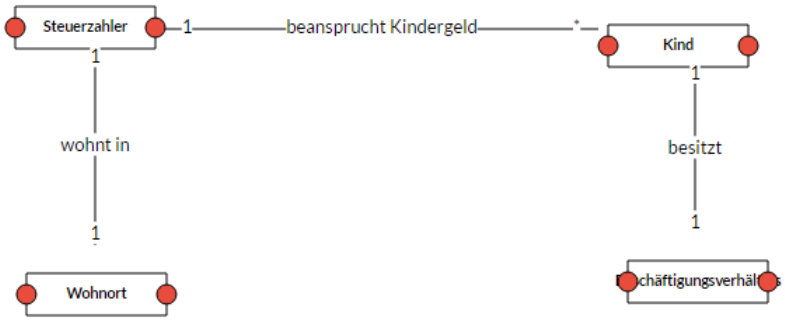
5.2 Evaluation of the case studies

The following section presents an evaluation in which the previously introduced case studies in Section 3.4 are implemented as semantic models in Lexia.

5.2.1 Child benefit

For the evaluation of child benefit regulations, the same semantic model as proposed in Section 3.4.1.3 was created with the graphical model editor. Figure 5.1 shows the resulting semantic model. The arithmetical and logical expressions had been transformed into valid MxL expressions. An example of an advanced expression, that had been used for the child benefit calculation, is shown in Figure 5.2. The first condition checks whether the child benefit claimer is not eligible for receiving it and the second part performs the actual calculation. Therefore, a function *betrag* expects a sequence of *Kind* objects and depending on the quantity the amount of child benefit is returned.

Figure 5.1: Semantic model of child benefit



Source: Own illustration

Figure 5.2: MxL expression for child benefit calculation

Source: Own illustration

Listing 5.1 shows the MxL expressions for *Steuerzahler*, Listing 5.2 the expressions for *Kind*. The set K^s (see Section 3.4.1.4) of all children of a taxpayer,

who are eligible for child benefit, are expressed with the query in Line 11 (Listing 5.1). The attribute *#betragFürKind* was abandoned in favor of a direct calculation in the *#summeKindergeld* attribute.

Listing 5.1: MxL expressions for type *Steuerzahler*

```
1  /* #istBerechtigt */
2  this.'wohnt in'.istImInland
3
4  /* #summeKindergeld */
5  if not this.'#istBerechtigt' then 0
6  else
7  let betrag = (kinder: Sequence) =>
8    if kinder.count(true) <=2 then kinder.count(true) * 190
9    else if kinder.count(true) = 3 then 2 * 190 + 1 * 196
10   else 2 * 190 + 1 * 196 + (kinder.count(true) - 3) * 221
11 in betrag(find(Kind).where(k => k.'#istBerechtigtesKind'))
```

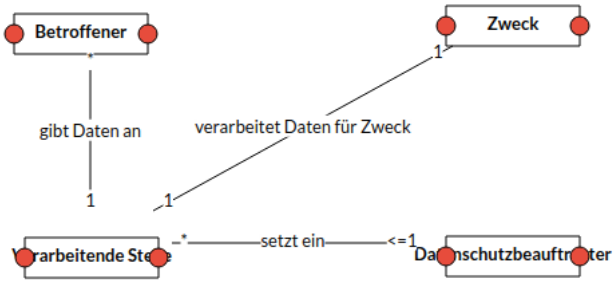
Listing 5.2: MxL expressions for type *Kind*

```
1  /* #alter */
2  (Today - this.geburtsDatum) / 365
3
4  /* #§32(4.1) */
5  this.'#alter' > 18 and this.'#alter' < 21 and not
       this.'besitzt'.inBeschäftigungsverhältnis and not this.'besitzt'.istArbeitssuchend
6
7  /* #§32(4.2) */
8  this.'#alter' > 18 and this.'#alter' < 25 and (this.'besitzt'.inBerufsausbildung or
       this.'besitzt'.inÜbergangszeit or this.'besitzt'.kannAusbildungNichtFortsetzen or
       this.'besitzt'.inFSJ)
9
10 /* #kind§32 */
11 this.'#§32(4.1)' or this.'#§32(4.2)' or this.istBehindert or this.'#alter' < 18
12
13 /* istBerechtigtesKind*/
14 (this.kindDesEheGatten or this.enkelKind or this.ersterGradVerwandt or
       this.pflegeKind) and this.'#kind§32'
```

5.2.2 Reporting obligation

For the reporting obligation, controlled by the *Bundesdatenschutzgesetz*, the same procedure as for the child benefit transposition was applied. The resulting semantic model is depicted in Figure 5.3.

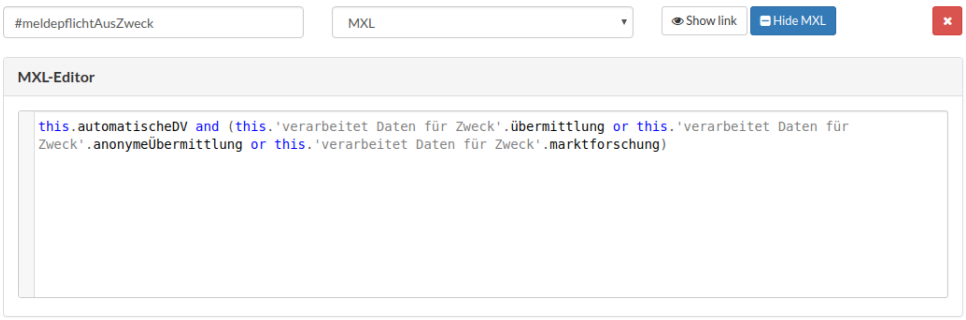
Figure 5.3: Semantic model for reporting obligation



Source: Own illustration

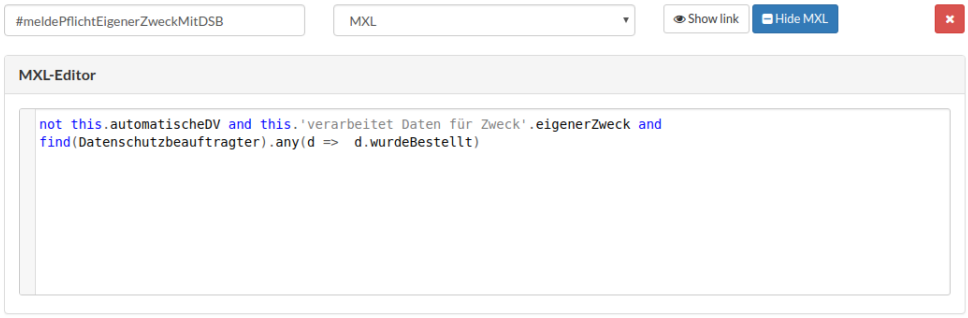
The two Figures 5.4 and 5.5 show two examples of the rules in Section 3.4.2.4 converted into MxL expressions. Each rule was transposed into a custom MxL expression and an additional disjunction checks whether one of this single rules is true (shown in Line 20 of Listing 5.3). The same approach was used for the prior checking (*vorabkontrolle*). Listing 5.3 shows all MxL expressions for the type *Verarbeitende Stelle*.

Figure 5.4: MxL expression of reporting obligation regarding purpose



Source: Own illustration

Figure 5.5: MxL expression of reporting obligation with data privacy officer



Source: Own illustration

Listing 5.3: MxL expressions for type *Verarbeitende Stelle*

```
1  /* #sonstigeDV */
2  not this.automatischeDV
3
4  /* #meldepflichtAusZweck */
5  this.automatischeDV and (this.'verarbeitet Daten für Zweck'.übermittlung or
       this.'verarbeitet Daten für Zweck'.anonymeÜbermittlung or this.'verarbeitet Daten
       für Zweck'.marktforschung)
6
7  /* #meldePflichtEigenerZweckMitDSB */
8  not (this.automatischeDV and this.'verarbeitet Daten für Zweck'.eigenerZweck and
       find(Datenschutzbeauftragter).any(d => d.wurdeBestellt))
9
10 /* meldePflichtEigenerZweckKeinDSBgr10Personen */
11 this.automatischeDV and this.'verarbeitet Daten für Zweck'.eigenerZweck and
       find(Datenschutzbeauftragter).any(d => not d.wurdeBestellt) and
       this.anzahlPersonen >= 10
12
13 /* #meldePflichtEigenerZweckKeinDSBkl10PersonenEinwilligung */
14 not (this.automatischeDV and this.'verarbeitet Daten für Zweck'.eigenerZweck and
       find(Datenschutzbeauftragter).any(d => not d.wurdeBestellt) and
       this.anzahlPersonen < 10and this.'gibt Daten an'.einwilligungVorhanden)
15
16 /* #meldePflichtEigenerZweckKeinDSBkl10PersonenVearbeitungErforderlich */
17 not (this.automatischeDV and this.'verarbeitet Daten für Zweck'.eigenerZweck and
       find(Datenschutzbeauftragter).any(d => not d.wurdeBestellt) and
       this.anzahlPersonen < 10and this.'verarbeitungErforderlich')
18
19 /* #meldepflicht */
```

```
20 this.'#meldepflichtAusZweck' or this.'#meldePflichtEigenerZweckMitDSB' or
       this.'#meldePflichtEigenerZweckKeinDSBgr10Personen' or
       this.'#meldePflichtEigenerZweckKeinDSBkl10PersonenEinwilligung' or
       this.'#meldePflichtEigenerZweckKeinDSBkl10PersonenVearbeitungErforderlich'
21
22 /* #vorabkontrolleWegenBesondererDaten */
23 this.automatischeDV and this.'gibt Daten an'.besondereArtPersonenbezogenerDaten and
       not this.verarbeitungErforderlich and not this.gesetzlicheVerpflichtung and not
       this.'gibt Daten an'.einwilligungVorhanden
24
25 /* #vorabkontrolleWegenBewertungDerPerson */
26 this.automatischeDV and this.'gibt Daten an'.bewertungDesBetroffenen and not
       this.verarbeitungErforderlich and not this.gesetzlicheVerpflichtung and not
       this.'gibt Daten an'.einwilligungVorhanden
27
28 /* #vorabkontrolle*/
29 this.'#vorabkontrolleWegenBesondererDaten' or
       this.'#vorabkontrolleWegenBewertungBetroffener'
```

5.3 Limitations

The following limitations have been identified during the implementation and usage of the modeling system.

Independence of attributes: Right now, all attributes are considered to be independent from each other. The model of the BDSG for reporting obligation can be used as an example: The entity *Zweck* has several boolean attributes, e.g. *eigenerZweck*, *übermittlung*, *anonymeÜbermittlung* and *marktforschung*. These attributes indicate the actual purpose for data processing and only one of them can be set to *true*. As a result the other attributes must be evaluated as *false*. The equations 5.1 to 5.4 show the constraints that must be fulfilled. However, in the current implementation an attribute can either be modeled to be set manually or to be evaluated automatically (as MxL expression).

$$eigenerZweck \implies \neg übermittlung \wedge \neg anonymeÜbermittlung \wedge \neg marktforschung \tag{5.1}$$

$$\begin{aligned} &\ddot{u}bermittlung \implies \\ &\quad \neg eigenerZweck \wedge \neg anonyme\ddot{U}bermittlung \wedge \neg marktforschung \end{aligned} \tag{5.2}$$

$$\begin{aligned} &anonyme\ddot{U}bermittlung \implies \\ &\quad \neg \ddot{u}bermittlung \wedge \neg eigenerZweck \wedge \neg marktforschung \end{aligned} \tag{5.3}$$

$$\begin{aligned} &marktforschung \implies \\ &\quad \neg \ddot{u}bermittlung \wedge \neg anonyme\ddot{U}bermittlung \wedge \neg eigenerZweck \end{aligned} \tag{5.4}$$

A solution could be the usage of a enumeration which is a supported data type by SocioCortex as well. However, this feature is currently not implemented in the modeling environment of Lexia. Instead of several "purpose attributes", only one attribute (*zweck*) could be created. This attribute has the data type *enumeration* and can only be set to one of the following values:

- EIGENER_ZWECK
- ÜBERMITTLUNG
- ANONYME_ÜBERMITTLUNG
- MARTKFORSCHUNG

Expressiveness of MxL: To this day, the expressiveness of the MxL language is still unknown. Therefore, it is possible that legal constructs exist which cannot be expressed in MxL. Examples could be semantics with non-monotonic logics or temporal logics.

Scalability of semantic model: In the current implementation, all data instances are directly stored in the semantic model. This has the advantage of neglecting the effort of joining data instances to the corresponding type structure. Since only data of one case at a time can be assigned to a semantic model, simultaneous evaluations of models cannot be executed. Afterwards,

all data must be wiped to enter new case data. Moreover, this approach lacks a persistent storage for previous cases.

A possible solution is the separation of instance data and type structure. In separate databases, the instance data can be stored along with a reference to a semantic model and its version. The versioning feature must be implemented in order to keep old case data valid, if a semantic model is changed and types are added or removed.

Workflow optimizations: The user interface of the modeling and the evaluation environment has some minor usability issues regarding the layout. For example as seen in Figure 5.1, the label of the type (*Beschäftigungsverhältnis*) is hardly readable, as there exists no overflow for long names in the *type* shapes. Moreover, the position of some buttons (*"Add attribute"* button in attribute modal dialog) can be improved.

Performance issues with large legal texts: If the user of the modeling environment has imported very large legal texts (such as the *Bürgerliches Gesetzbuch BGB*), the performance of the front end application drops significantly. It is assumed that the performance drops are the result of the Angular.js digest cycle, performing a "dirty checking" on scope properties. Since the legal text consists of an array of paragraphs which are rendered by the *ngRepeat* directive, a watcher is created for each entry of this array. If a legal text consists of several hundreds or even thousands of paragraphs, the same amount of two-way-data-binded scope properties are created that must be "dirty checked" in every digest cycle. This is of course costly in terms of time and is absolutely not necessary, since the legal text is static and not intended to be altered.

6 Summary, outlook and conclusion

6.1 Summary

In the framework of this master's thesis, a prototypical modeling environment was implemented in Lexia, the data science environment for analyzing legal texts from the chair of *Software Engineering for Business Information Systems* of the TU München. This modeling environment allows users to create *semantic models* from legal texts which capture formally the defined semantics in these texts. Furthermore, the semantic model provides support for basic legal reasoning and can be used for the evaluation of (hypothetical) cases.

At first, a brief introduction was given concluding recent evolutions in the domain of artificial intelligence and law. Next, the research questions for this thesis were proposed. Afterwards, a chapter summarizing the related work was introduced. In this chapter, the benefits of formalizing semantics defined in laws were highlighted as well as previous approaches of knowledge representations were introduced. It concludes with a short overview of the functionality of legal expert systems including some examples of such systems.

In the context of the analysis chapter, the involved systems were introduced, namely *Lexia* and *SocioCortex*. Lexia is a legal data analysis platform that can be used for the exploration of legal documents as well as for the identification of linguistic patterns. SocioCortex is a hybrid wiki with support for a powerful expression language. In the implementation phase of this thesis, Lexia was extended by a modeling environment which allows the creation of semantic models including static and dynamic attributes and relations between the modeled entities. These models are then transposed into SocioCortex that stores these models as well as performs reasoning through the evaluation of the defined MxL expressions.

For a later evaluation, two case studies were presented. The child benefit from the *Einkommenssteuergesetz* and the reporting obligation of the *Bundesdaten-*

schutzgesetz are two concrete examples of normative regulations that were later formalized in Lexia. Both case studies were analyzed regarding their decision structure resulting in a formal definition of the semantics. Target models, which are used as blueprint for the actual semantic models, were created as well. The analysis section also emphasizes the different stakeholders of the system as well as a requirements analysis.

The next section puts focus on the actual implementation in Lexia. It describes the adjustments in the data model and in the REST client, used for communication between Lexia and SocioCortex. The creation of the complete user interface with Angular.js is elucidated as well. This section also presents the structure of semantic models, serialized in JSON, which is the data exchange format between front end and back end. Furthermore, another important part were the explanations about the information extraction process from the visual model.

In the framework of the evaluation, the research questions of this master thesis are answered and the results of the cases studies are presented. The evaluation reflects the work of this master thesis and shows the identified limitations of the modeling system.

The very last chapter of this thesis deals with a summary, a short outlook of improvements for the modeling environment as well as new trends in decision modeling.

6.2 Outlook

This section provides an outlook of how the development of the modeling system in Lexia can be continued as well as recent trends in decision modeling.

6.2.1 Improvements for modeling environment in Lexia

A first improvement for the modeling environment could be the usage of comments. It would be helpful to assign comments to different semantic model elements, in order to help achieving the same level of comprehensibility between the different legal data scientists working on a model. Moreover, users applying the semantic models would benefit from comments, if they provide

additional information for instance how norms should be applied. The in Section 2.3.3.2 introduced *knowledgeTools* system already includes such a message board, allowing to draft comments for nodes of the knowledge tree.

A next improvement could be the highlighting of all annotations in the text during the modeling phase. Right now, the annotations can only be used through a drop down box in views using the *link* directive (see Section 4.2.2.3), which lacks of an enhanced overview. The distribution of annotations in the text and the different types of annotations can be supportive for the formalization process.

In the evaluation view, the linked text for *types* and *attributes* can be exposed, but this feature is still missing for *relations*. Therefore, the user interface must be enhanced so that the linked text for relations can also be shown on request.

The last improvement would provide the major benefit increment for the modeling system. If a proper semantic model has been created and also the links between model elements and text have been established, a new component could analyze the impact of importing a new document version. This component must expose which sections of a document have been changed compared to its previous version. If changes are made at textual units, that are linked with semantic model elements, a message should appear and request checking the validity of the modeled semantics for this element.

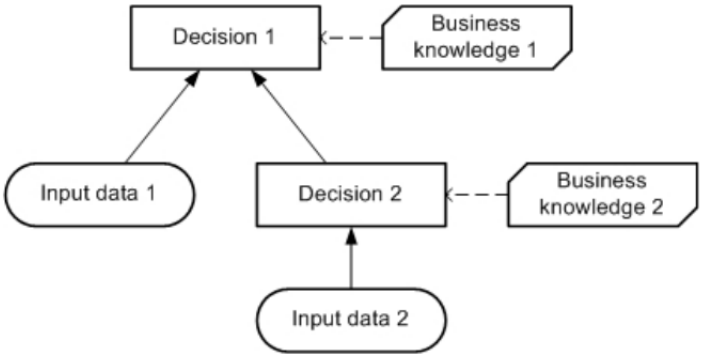
6.2.2 Latest trends in decision modeling

The *Decision Model Notation (DMN)* is a fairly new standard[23] for modeling decisions, published by the *Object Modeling Group*. Scope of DMG is to model decisions by business analysts that are performed in day-to-day business processes. This specification [18] is used to "model human-decision making", for "modeling requirements for automated decision-making" and also for "implementing automated decision making". Therefore, it is also suitable for modeling and executing legal decisions.

Currently, decision models can be described by the following two standards: *Business Process Models* (BPMN), that define tasks and activities that are

[23] The version 1.1 was released in June 2016.

Figure 6.1: Example of a *Decision Requirements Diagram*



Source: [18, p. 21]

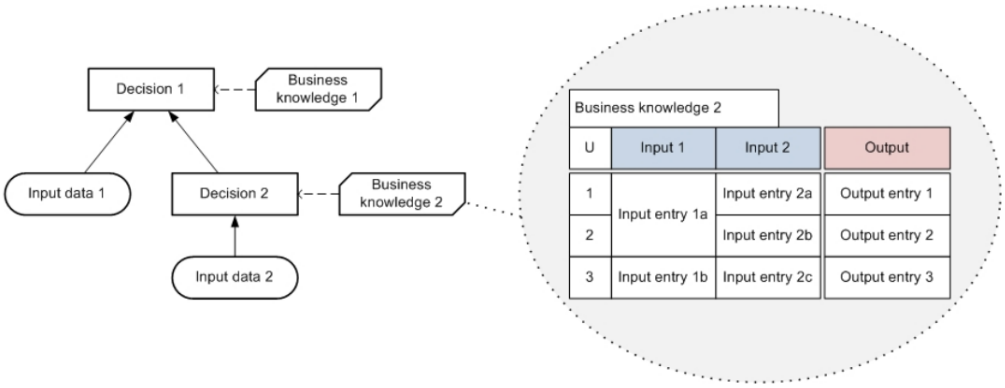
used for decision making, and *Decision Logics* which are used for describing individual decisions by business rules or decision tables. The new proposed DMN works as a link between *BPMN* and *Decision logic* in order to connect these standards. DMN specifies three different concepts which contain several artifacts:

Decision requirements level: On the requirements level it is necessary for business analysts to identify business decisions, their relationship to each other, the data necessary for making decisions and also the sources of business knowledge. An artifact of the DMN specification on this level is a *Decision Requirements Graph (DRG)* which consists of one or many *Decision Requirements Diagrams (DRD)*. A DRD depicts how the different requirements depend on each other, on input data and on business knowledge models. The business knowledge models encapsulate business know-how in form of analytic models or business rules. Figure 6.1 shows an example of a simple *Decision Requirements Graph* containing two decisions. In this depicted situation, *Decision 1* requires the output of *Decision 2* [18, p. 20ff].

Decision logic level: At this level, the components defined at the *Decision requirements level* are specified even further to capture the total set of business rules and calculations which can then be used for automated decision-making. A representation of these logics is for example the *Decision Table*. This table encodes a mapping of a set of inputs to a set of outputs expressed in rules.

Figure 6.2 shows an example of such a table. Alternatively, DMN specifies an expression language which is called *Friendly Enough Expression Language (FEEL)*. At this point it is not further investigated how this language would compete with the MxL [18, p. 22ff] .

Figure 6.2: Example of a *Decision Table*



Source: [18, p. 24]

Decision service: In order to provide automated decision-making, the decision-making logic must be organized in *Decision services*. Such a service includes one or more decisions from a decision model and exposes a service interface that can be consumed by a task in a BPMN model. For invoking a service, mandatory input data must be provided and a decision result is returned. The services are stateless and do not have any side effects. An implementation of such a service could be realized as a web-service. DMN does not specify the actual implementation method, but only the functionality against a decision model [18, p. 25].

6.3 Conclusion

This master's thesis contributes to the formalization of semantics defined in legal texts with the help of a newly created modeling component in Lexia. As claimed at the beginning, to the first step of transposing norms in unstructured text to a formal model were paid less attention than to the second step of automating the law.

The solution created in this work provides the necessary tool support to fulfill this task in a convenient way, allowing multiple legal experts to share their knowledge and create a well-defined model in a collaborative way. The reference to the text, which is the base for semantic models, is always present. With a strict adherence of the isomorphism concept, every model element is linked with at least one reference in the source text. This feature is not only helpful for the users itself, but also for an impact analysis of changing norms on semantic models.

The semantic model has similarities to ontologies, but instead of any kind of logics used for inference, this approach uses a rule-based expression language. These rules cannot only express boolean or logical terms, but also arithmetical functions. It is considered that this approach is more expressive than description logics.

The tool implementation presented in this thesis stands in competition with other research approaches and commercial products. Moreover, a new standard in decision modeling arises which will be implemented in several tools sooner or later. Right now, there exists no user feedback for the modeling environment implemented in this thesis. Since it fulfills all necessary requirements for creating a formal and visual representation of normative texts, the modeling component provides a valuable contribution to Lexia.

Bibliography

[1] Knowledge Tool Case Management Screencaset. https://www.knowledgetools.de/fallmanagement_video.html. Accessed: 2016-09-21.

[2] Oracle Policy Automation. http://documentation.custhelp.com/euf/assets/devdocs/august2016/PolicyAutomation/en/Default.htm. Accessed: 2016-09-10.

[3] OWL 2 Web Ontology Language Document Overview (Second Edition). https://www.w3.org/TR/owl2-overview/. Accessed: 2016-09-05.

[4] RDF 1.1 Turtle. https://www.w3.org/TR/turtle/. Accessed: 2016-09-05.

[5] RDF 1.1 XML Syntax. https://www.w3.org/TR/rdf-syntax-grammar/. Accessed: 2016-09-05.

[6] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59, March 1994.

[7] F. Baader, I. Horrocks, and U. Sattler. Description logics. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks in Information Systems, pages 3–28. Springer–Verlag, Berlin, Germany, 2004.

[8] F. Baader and U. Sattler. Description logics with concrete domains and aggregation. In H. Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 336–340. John Wiley & Sons Ltd, 1998.

[9] T. J. M. Bench-Capon and F. P. Coenen. Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, 1(1):65–86, 1992.

[10] Trevor Bench-Capon, Michał Araszkiewicz, Kevin Ashley, Katie Atkinson, Floris Bex, Filipe Borges, Daniele Bourcier, Paul Bourgine, Jack G. Conrad, Enrico Francesconi, Thomas F. Gordon, Guido Governatori, Jochen L. Leidner, David D. Lewis, Ronald P. Loui, L. Thorne McCarty, Henry Prakken, Frank Schilder, Erich Schweighofer, Paul Thompson, Alex Tyrrell, Bart Verheij, Douglas N. Walton, and Adam Z. Wyner. A history of ai and law in 50 papers: 25 years of the international conference on ai and law. *Artificial Intelligence and Law*, 20(3):215–319, 2012.

[11] Marco Brattinga and Sjir Nijssen. *On the Move to Meaningful Internet Systems: OTM 2015 Workshops: Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, EI2N, FBM, INBAST, ISDE, META4eS, and MSC 2015, Rhodes, Greece, October 26-30, 2015. Proceedings*, chapter A Sustainable Architecture for Durable Modeling of Laws and Regulations and Main Concepts of the Durable Model, pages 254–265. Springer International Publishing, Cham, 2015.

[12] Stephan Breidenbach. Landkarten des Rechts - von den Chancen industrieller Rechtsdienstleistungen. https://knowledgetools.de/download/Umbr1_Breidenbach.pdf, 2009. Accessed: 2016-09-22.

[13] Bundesrepublik Deutschland. Bundesdatenschutzgesetz (bdsg), 2016. Accessed on 2016-07-04.

[14] Bundesrepublik Deutschland. Einkommenssteuergesetz (estg), 2016. Accessed on 2016-07-04.

[15] Tom Engers and Sjir Nijssen. *Electronic Government: 13th IFIP WG 8.5 International Conference, EGOV 2014, Dublin, Ireland, September 1-3, 2014. Proceedings*, chapter Connecting People: Semantic-Conceptual Modeling for Laws and Regulations, pages 133–146. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[17] Thomas F. Gordon. From jhering to alexy – using artificial intelligence models in jurisprudence, 1994.

[18] Object Management Group. Decision Model and Notation. http://www.omg.org/spec/DMN/1.1/PDF, 2016. Accessed: 2016-09-20.

[19] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[20] C. D. Hafner. Conceptual organization of case law knowledge bases. In *Proceedings of the 1st International Conference on Artificial Intelligence and Law*, ICAIL '87, pages 35–42, New York, NY, USA, 1987. ACM.

[21] Wesley Newcomb Hohfeld. Some fundamental legal conceptions as applied in judicial reasoning. *The Yale Law Journal*, 23(1):16–59, 1913.

[22] H. Honsell and T. Mayer-Maly. *Rechtswissenschaft: Eine Einführung in das Recht und seine Grundlagen*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2015.

[23] Thomas Jandach. *Juristische Expertensysteme*. Springer Berlin Heidelberg, 1993.

[24] Florian Matthes, Christian Neubert, and Alexander Steinhoff. Hybrid wikis: Empowering users to collaboratively structure information. In *ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies, Volume 1, Seville, Spain, 18-21 July, 2011*, pages 250–259, 2011.

[25] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[26] I. Monahov, T. Reschenhofer, and F. Matthes. Design and prototypical implementation of a language empowering business users to define key performance indicators for enterprise architecture management. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 337–346, Sept 2013.

[27] James Popple. A pragmatic legal expert system. *APPLIED LEGAL PHILOSOPHY SERIES, Dartmouth (Ashgate), Aldershot*, 1996.

[28] F. Puppe. *Einführung in Expertensysteme*. Studienreihe Informatik. Springer Berlin Heidelberg, 2013.

[29] Oliver Raabe, Christian Baumann, Christian Funk, Daniel Oberle, and Richard Wacker. *Recht ex machina : Formalisierung des Rechts im Internet der Dienste*. Imprint: Springer, Berlin, Heidelberg ;s.l, 2012.

[30] Thomas Reschenhofer. Design and prototypical implementation of a model-based structure for the definition and calculation of Enterprise Architecture Key Performance Indicators. Master's thesis, Technische Universität München, Germany, 2013.

[31] Thomas Reschenhofer, Manoj Bhat, Adrian Hernandez-Mendez, and Florian Matthes. Lessons learned in aligning data and model evolution in collaborative information systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 132–141, New York, NY, USA, 2016. ACM.

[32] E. L. Rissland and K. D. Ashley. A case-based system for trade secrets law. In *Proceedings of the 1st International Conference on Artificial Intelligence and Law*, ICAIL '87, pages 60–66, New York, NY, USA, 1987. ACM.

[33] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[34] Johannes Scharf. *Künstliche Intelligenz und Recht: Von der Wissenrepräsentation zur automatisierten Entscheidungsfindung*. Österreichische Computer Gesellschaft, 2015.

[35] Matthias Schmid. Einblick in die moderne werkstatt der gesetzgebung. In K.M. Eichhoff-Cyrus and G. Antos, editors, *Verständlichkeit als Bürgerrecht?: Die Rechts- und Verwaltungssprache in der öffentlichen Diskussion*, Duden - Thema Deutsch, pages 244–254. Bibliographisches Institut, 2014.

[36] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, May 1986.

[37] Tom M. van Engers and Robert van Doesburg. At your service, on the definition of services from sources of law. In *Proceedings of the 15th Inter-*

national Conference on Artificial Intelligence and Law, ICAIL '15, pages 221–225, New York, NY, USA, 2015. ACM.

[38] Tom Maarten van Engers and Robert van Doesburg, editors. *First steps towards a formal analysis of law*, eKNOW 2015. IARIA XPS Press, 2015.

[39] Frank van Harmelen, Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of Knowledge Representation*. Elsevier Science, San Diego, USA, 2007.

[40] Bernhard Waltl and Corinna Coupette. Oral talk about requirements on june, 29th. 2016, June 2016.

[41] Bernhard Waltl, Florian Matthes, Tobias Waltl, and Thomas Grass. Lexia - a data science environment for semantic analysis of german legal texts. In *19. Internationales Rechtsinformatik Symposium*, Salzburg, Austria, 2016.