Modeling, Execution and Analysis of Formalized Legal Norms in Model Based Decision Structures

Bernhard Waltl, Thomas Reschenhofer, and Florian Matthes

Software Engineering for Business Information Systems
Department of Informatics
Technische Universität München
Boltzmannstr. 3, 85748 Garching bei München, Germany
b.waltl@tum.de, reschenh@in.tum.de, matthes@in.tum.de

Abstract. This paper describes a decision support system to represent the semantics of legal norms. The focus is on designing and software-technical implementing of a comprehensive system to support model based reasoning on legal norms and enabling end-users to create, maintain, and analyze semantic models, i.e. ontologies, representing structure and semantics of norms.
A model based expression language (MxL) has been developed to coherently support the formalization of logical and arithmetical operations. MxL is intended to define complex, nested, strongly-typed, and functional operations. The paper summarizes research on the design and implementation of a legal expert system built upon model based decision structures. Thereby, three different components, namely a model store, a model execution component, and an interaction component have been developed. The formalization, execution, and analysis is shown on German child benefit regulations.

Keywords: Legal expert system, deductive reasoning, rule-based system, end-user centered, domain specific language

1 Introduction

The formalization of normative texts, regarding to various aspects, e.g., propositional, deontological, temporal, defeasible, etc., is well studied [1–3] and led to valuable principles allowing complex formal reasoning on given laws or contracts. The execution of algorithmically processable formalizations requires the translation of the textual representation, which is a complex and manual task [4]. Advances in natural language processing have at most led to tools or algorithms supporting end-users during analysis, interpretation, and application of legal rules [5]. The user perspective during the formalization process of legal rules has earned rather less attention within the last decades [6]. Higher attention has user-enabled formalization of arguments drawn [7].

The paper presents selected related approaches and ontologies for legal reasoning in Section 2. In Section 3 the model based formalization is introduced and illustrated on a concrete example from the German tax law. The design and implementation with a strong focus on end-user perspective of the resulting decision support system is summarized in Section 4. The automated analysis of decision structures and their representation are presented in Section 5. Finally critical and concluding remarks are discussed in Section 6.

2 Related Work

Within the last decades many attempts have been made to formalize legal systems, respectively legal rules. Thereby, the contributions made by the AI and law community have significantly improved the understanding of the possibilities and limitations of formalization [1].

2.1 A Quick Glance at Legal Expert Systems

Legal expert systems (LES) are well-studied throughout the domain of artificial intelligence and law [1]. Highly tailored to the legal domain the usage of those systems was left up to experts with knowledge in both legal sciences and computer science.

The reasoning within legal expert systems, can be differentiated into several categories, such as rule-based systems (e.g., [7]), case-based reasoning (e.g., [8]), neural nets (e.g., [9]), fuzzy logic (e.g., [10]), Bayesian networks (e.g., [11]), etc.. An additional step towards more comprehensive LES was reasoning on (legal) ontologies [12–14]. Expressing logical constraints and relationships between those knowledge objects can be done with description logics, e.g., web ontology language (OWL). Thereby, OWL allows the definition of constraints and axioms in ontologies. Many prior attempts used the W3C standard for modeling ontologies based on RDF and OWL. OWL suffers of the possibility to formalize arithmetical or complex logic operations. Since OWL is an description logic, it was not designed to be used for arithmetical expressions or as first-order (or higher order) predicate language.

Latest approaches on the formalization of legal norms using LegalRuleML have investigated structured processes to move from a natural language to a controlled natural language focusing on the pragmatics of legal reasoning [15]. In the model proposed by Ramakrishna they need two roles, namely legal practitioner and a knowledge engineer, to formalize the semantics in an independent rule representation format, namely LegalRuleML. Another very interesting approach was implemented by [16]. The authors focused on the implementation of a rule management architecture integrating a defeasible inference engine.

2.2 End-User Oriented Decision and Reasoning Systems

Enabling end-users to understand, analyze, and model use cases, which are relevant to them, e.g., user stories, has become an important paradigm in the

domains of software engineering. Thereby, the focus is on providing modeling languages, i.e. notations, that are expressive enough to capture all relevant issues of a particular domain while remaining simple enough to be used by end-users. Well-known examples of those end-user oriented modeling languages are UML (Unified Modeling Language [17]), BPMN (Business Process Modeling Notation [18]), CMMN (Case Management Modeling Notation [19]), or DMN (Decision Model and Notation [20]).

BPMN focuses on business modeling and specification [18]. Many attempts have already been made to adapt the BPMN to enable automated compliance checks regarding pre-defined executable rules [21]. The most recent standards provided by the OMG (Object Management Group) have a more specific focus on supporting modeling decision structures in work flows, e.g. business processes or adaptive cases. Both, the CMMN and the DMN, provide end-users with functionality to specify complex decision structures, such as decision tables [20, Clause 8].

This brief sketch of the development shows how important the end-users orientation and empowerment has been for the success of modeling notations. It also shows, that the provision of executable semantics has always been an important part of the standardizations efforts. Within the last years more and more effort has been spent on formalizing work flows and decision structures. This heavily increases the transparency of business processes and adaptive cases and allows optimizations and leveraging of efficiency and effectiveness.

3 Model Based Formalization Of Normative Regulations

3.1 The German Child Benefit Regulation

The German child benefit regulation is part of the tax law and can be formalized using an ontological, i.e., model-based approach. The relevant articles from the law are German tax income act §32, and §§62–78. §32 legally defines the term "child" and what attributes are necessary for human beings to be considered as children regarding the German tax law. We expressed the semantics of the types, attributes and the relations in a UML class diagram (see Figure 1).

The class diagram represents the user-defined semantic model. The result of this model is still the result of a manual interpretation process, which can be supported by analytics from the NLP framework, e.g., [22], to automatically classify legal norms, such as obligations and permissions.

The semantic model consists of four different types, namely *Taxpayer*, *Residence*, *Child*, and *Employment*. Each type has atomic attributes, indicated by '-', and derived attributes, indicated by '/'. For the purpose of this model, calculation of child benefit, it is sufficient for the *Taxpayer* to have only one atomic attribute *name* of type string. In addition, the *Taxpayer* has two derived attributes *isQualified* and *sumChildbenefit*. Both are defined through a MxL expression, which is a strongly typed domain specific model based expression language (see Sections 3.2 and 4.1).

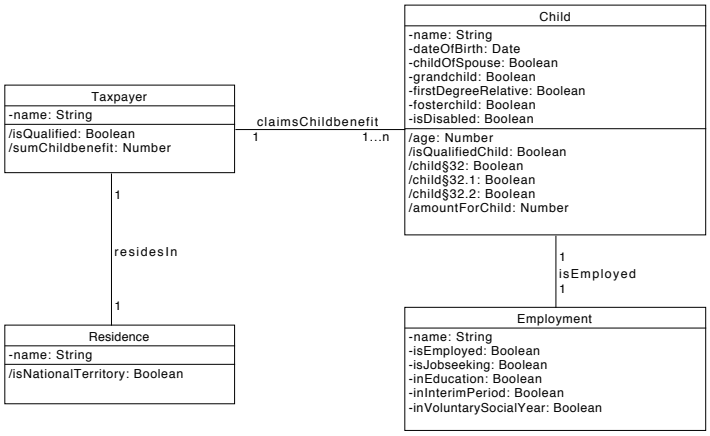

Fig. 1. A semantic model for the German child benefit claim.

3.2 Model Based Formalization

We show the appropriateness of our approach by a concrete example from the German tax law, namely the determination if a taxpayer is eligible for retrieving child benefit and the calculation of child benefit. Different conditions have to be fulfilled, which are stated in German tax law.

Type definitions For the determination of the child benefit the German tax law requires several different types, namely taxpayer, child, residence, and employment.

$$t \in taxpayer \tag{1}$$

$$c_j \in child,\ j \in \mathbb{N} \tag{2}$$

$$r \in residence \tag{3}$$

$$e \in employment \tag{4}$$

Relations The types have relations among each other, which have to be formalized. Thereby, we can restrict our model to three different relations (see Figure 1). The relation between a taxpayer and its child (5), between a taxpayer and its residence (6), and finally between a child and its employment (7).

$$claimsChildBenefit \subseteq taxpayer \times child : \\ (t, c_j) \in claimsChildBenefit \implies k_j \; is \; t's \; j^{th} \; child \tag{5}$$

$$residesIn \subseteq taxpayer \times residence : \\ (s, w) \in residesIn \implies taxpayer \; t \; lives \; in \; r \tag{6}$$

$$isEmployed \subseteq child \times employment : \\ (c_j, e) \in isEmployed \implies child \; c_j \; is \; employed \tag{7}$$

Derived Attributes & Rules Finally, the different types have attributes, such as name, birth date, etc., that are required during the reasoning process. Those attributes can either be atomic attributes or derived attributes. The first kind of attributes describe those that inherently belong to a concrete type (e.g., name is of type string). The latter ones are those attributes that can be inferred from other attributes. Consequently, derived attributes consist of the definition, i.e. expression, containing the required information on how the attributes is determined.

The following equations describe how the different attributes, which are required during the determination of child benefit, are defined. The notation for accessing an attribute of a type is the '.' (dot). E.g., $c_j.dateOfBirth$ means the dateOfBirth attribute from the j^{th} child.

$$c_j.age := \lfloor NOW - c_j.dateOfBirth \rfloor \tag{8}$$

$$\begin{aligned} c_j.isQualifiedChild := \; & c_j.child\S 32 \wedge \\ & (c_j.childOfSpouse \\ & \vee \, c_j.grandchild \\ & \vee \, c_j.firstDegreeRelative \\ & \vee \, c_j.fosterchild) \end{aligned} \tag{9}$$

$$\begin{aligned} c_j.child\S 32 := \; & c_j.age < 18 \\ & \vee \, c_j.isDisabled \\ & \vee \, c_j.\S 32.4.1 \\ & \vee \, c_j.\S 32.4.2 \end{aligned} \tag{10}$$

$$\begin{aligned} c_j.\S 32.4.1 := \; & (c_j.age > 18 \wedge c_j.age < 21) \\ & \wedge \neg e.isEmployed \\ & \wedge \neg e.isJobseeking \end{aligned} \tag{11}$$

$$
\begin{aligned}
c_j.\S32.4.2 := (c_j.age > 18 \wedge c_j.age < 25) \\
\wedge (e.inEducation \\
\vee e.inInterimPeriod \\
\vee e.inVoluntarySocialYear)
\end{aligned} \tag{12}
$$

The equations (9) – (12) specify the different conditions that are defined by law qualifying a child to be considered during the calculation for child benefit. The claim can arise from different articles from the tax law. For example §32 states that a child, which is younger than 18 or disabled is eligible. Beside, it is also considered as child if the conditions in §32.4.1 or §32.4.2 are met.

$$
\begin{aligned}
C^t := \{c_j \in child | (t, c_j) \in claimsChildbenefit \\
\wedge\ c_j.isQualifiedChild\}\ for\ t \in taxpayer
\end{aligned} \tag{13}
$$

$$
t.isQualified :=\ r.isNationalTerritory,\ (t, r) \in livesIn \tag{14}
$$

The equations (13) and (14) ensure that a child is related to a taxpayer and that the taxpayer lives within the national territory.

$$
t.sumChildbenefit :=
\begin{cases}
\sum_{c_j \in C^t} c_j.amountForChild & if\ t.isQualified \\
0 & if\ \neg t.isQualified
\end{cases} \tag{15}
$$

$$
c_j.amountForChild := \begin{cases}
190 & if\ 1 \leq j \leq 2 \\
196 & if\ 3 \leq j \leq 4 \\
221 & if\ 5 \leq j
\end{cases} \tag{16}
$$

The remaining two equations (15) and (16) determine the amount for the child benefit based on the number of eligible children.

MxL - A model based expression language The model based expression language (MxL) as a type-safe domain specific language was developed in our research group (see [23]) to support reasoning within a generic meta-model based information system.

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

An important property of MxL is its type-safety: It ensures that expressions are valid regarding their static semantics and thus supports users in defining

consistent expressions with respect to the user-defined semantic model. Furthermore, MxL's type-safety enables the system to resolve dependencies between expressions, which in turn enables an automated adoption of expressions if referenced elements of the semantic model change.

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

4 The Model Based Legal Expert System

4.1 Executable semantics of legal norms

To represent the content of legal norms in model based decision structures two aspects need to be differentiated to fully capture the executable semantics: Firstly, the structural semantics of legal norms (see Section 4.1). Secondly, the behavioral semantics of legal norms (see Section 4.1).

Modeling structural semantics of legal norms Modeling structural properties of the norms' content is necessary to capture mandatory legal concepts with their attributes and relations. The meta model based information system [23] empowers end-users to iteratively and collaboratively define semantic models. End-users create and manage domain-specific types as well as corresponding attributes and relations, e.g., a type *Taxpayer* with a relation *residesIn*, or another type *Child* with the attribute *dateOfBirth* and the derived attribute *age*. Since the semantic model is stored in a collaborative environment, i.e. service oriented architecture for the model store, it can be revised by other users. They can refine the model by adding additional types, attributes or relations. They could also remove or add different kinds of constraints, e.g., cardinality constraints (the number of minimal and/or maximal values an attribute has to have), or type- and structure-related constraints, e.g., data types of attributes. As illustrated in Figure 1, the cardinality of the *Taxpayer*'s relation *Child* is defined to be *1...n*, which allows a taxpayer to claim child benefit for different children.

Modeling behavioral semantics of legal norms Besides the semantic structure of legal concepts the behavioral aspects focus is on the interconnection and the reasoning dependencies within structural components, namely types, attributes, and relationships.

The listing below shows the formalization of the normative structure deciding on whether a child is eligible for child benefit or not. Thereby, the function uses logical and arithmetical operations to combine numeric (e. g., *greater-than*) and boolean attributes (e.g., *or, and, not*) of the child. It corresponds to the derived attribute definition of Equation 11 in Section 3.

Custom MxL Function Child::§32.4.1

Description	Based on the attributes of a child this function determines whether the child is eligible for a claim regarding §32.4.1 or not.
Parameters	None
Return Type	Boolean

Expression	`(this.'#age' > 18.0 and this.'#age' < 21.0)`
	`and`
	`(not this.'hasEmployment'.isEmployed or`
	`not this.'hasEmployment'.isJobseeking)`

Fig. 2. Definition of the function *§32.4.1.* A short description, the input parameters, the return type, and the MxL expression are provided.

```
(this.'#age' > 18.0 and this.'#age' < 21.0)
 and
(not this.'hasEmployment'.isEmployed or
 not this.'hasEmployment'.isJobseeking)
```

The decision structure belongs to a child instance, which can be accessed using the *this* keyword. The semantics of the keyword is the same as in object oriented programming languages, namely accessing the attributes or methods from the same instance. Consequently, if the value of the derived attribute c_j.§32.4.1, $c_j \in child,\ j \in \mathbb{N}$ is read, the decision structure above is evaluated and the result calculated.

We can combine and nest the functions to define a function *sumChildbenefit* (see below) determining how much a given taxpayer t receives. The function reflects the Equations 13 (line 7), 15 (line 1 and 7), and 16 (line 4-6) of Section 3.

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

If a given taxpayer *tp* is not eligible, the function returns 0. Otherwise, the function determines the number of children for which the taxpayer can claim the benefit. Based on this result, the function calculates the amount that a taxpayer will receive based on a formal calculation prescription derived from the law. This small example demonstrates how partial expressions can be defined as separate custom functions in order to formalize specific norms. It shows how to apply and combine different kind of operations, e.g., conditionals (if-then-else) or logical (and, or, not, etc.) and arithmetic (greater-than, summation, multiplication, etc.) operations, in order to formalize behavioral aspects of normative texts. In addition it shows the capabilities of MxL as a query language, that allows complex queries to retrieve instances fulfilling a particular condition, such as all

children having the attribute *isQualifiedChild* set on true (see line 7 in listing above).

4.2 Software Components of the Model Based Legal Expert System

Jandach [24] analyzed different notions of legal expert systems in 1993 with a particular focus the concepts and characteristics that address LES for civil law systems, more specifically the legal system in Germany. Several attempts have been made to implement decision structures, arising from German legal texts, into rule-based systems. However, hardly any attempt has been made to formalize German laws using a model based, i.e., ontological approach, with a reasoning engine that enables users to define expressions and infer knowledge using propositional logic, first-order predicate logic, and arithmetical logic alike.

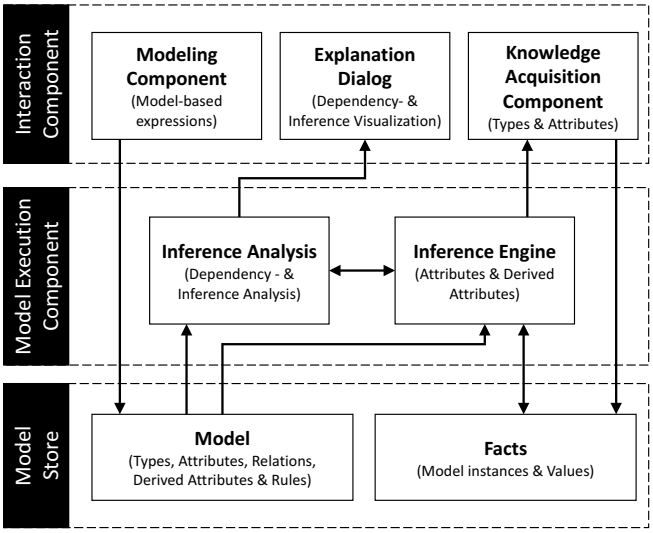

Fig. 3. System architecture consisting of a model storage, model execution components, and interaction components.

Based on Jandachs classification of the different components of legal expert systems, namely knowledge base, inference engine, explanation component, knowledge caption component, and dialog component, we designed a software system, which's components are shown in Figure 3.

The system's components can be classified into three different groups, namely a model store, a model execution component, and an interaction component. Each group is implemented by multiple different software components.

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

Model A model is described by its types with their attributes, i.e., scheme, and the relations between types. Our implementation differentiates between two types of attributes: atomic attributes and so call derived attributes. Atomic attributes consist of concrete values and have a basic data type, such as number, date, text, enumeration, boolean, and sequence. In contrary, derived attributes are expressed as rules, formalized in a model based expression language (MxL), which is a strongly-typed and functional domain specific language (DSL).

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

For our implementation we used a meta-model based information system to implement the model storage component. This meta-model based information system holds all the information about the scheme of the ontology, i.e., model, as well as the instances (facts) of a concrete model. The system allows the formalization of different models, that are logically separated into disjoint workspaces (data rooms).

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

Inference Engine The reasoning on the given facts considering the formalized rules requires access to the database of facts and the storage holding the information about the expressions required to determine the derived attributes. The input parameters and the return values of those expressions are strongly typed. The inference engine is developed in Java and retrieves data from the meta model based information system. The MxL was designed using "Beaver - a LALR Parser Generator"[1]. The inference engine offers end-users to define semantics of derived attributes in functional expressions, and allows expression of first and second order logic as well as the definition of complex queries (projection, selection, and transformation).

Inference Analysis Closely connected to the inference engine is the inference analysis component. This component allows the inspection of complex expressions. On the one hand it is possible to get the information about the abstract syntax tree (AST) of an expression. It allows overviews on the provided and derived facts in an object diagram like visualization (see Section 5.1). On the other hand the component offers functionality to view complex data flows based on the input parameters to inspect the resulting derived attribute (see Section 5.2).

[1] http://beaver.sourceforge.net/, last access on 03/01/17

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

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

Modeling Component Users, e.g., legal data scientists or legal knowledge engineers, are supported during the creation of the model with its attributes and relations by an appropriate modeling component. The modeling component offers functionality to add new types to a model and easily add attribute definitions and derived attribute definitions (MxL). Thereby, the users are supported with a graphical user interface that runs completely as web application.

Knowledge Acquisition Component The insertion of facts is done in a separate component, which has been exclusively designed for this purpose. Thereby, types and attributes, describing a case, can be added by the users. Since the model attributes are typed a first type checking is performed within the interface. It is not possible to insert not compatible types, such as text if number is required. The knowledge acquisition components show the inferred values of the derived attributes.

Explanation Dialog To reconstruct the conclusion that was made by the system, users need functionality providing them with information about the reasoning procedures. This explanation dialog visualizes the information from the inference analysis component. Thereby, given a particular derived attribute the user gets information about the underlying MxL expression and the abstract syntax tree showing the different data values and operators that contribute to the overall result.

4.3 Execution and computation of models in a collaborative environment

The three interaction components of our system, namely modeling component, explanation dialog and the knowledge acquisition component (see Figure 3), are developed as collaborative web application components. Each of them can be used through a web browser. They access a shared model store containing the model and the instances via a REST API. This enables users to create new models, i.e., types, attributes, and relations, and share them with other users. Additionally, it is possible to maintain and refine existing models that a user has created prior or that another user created.

The creation of decision structures through the system components is separated from the components handling the instantiation of models and their execution. The knowledge acquisition component, shown in Figure 4, is structured

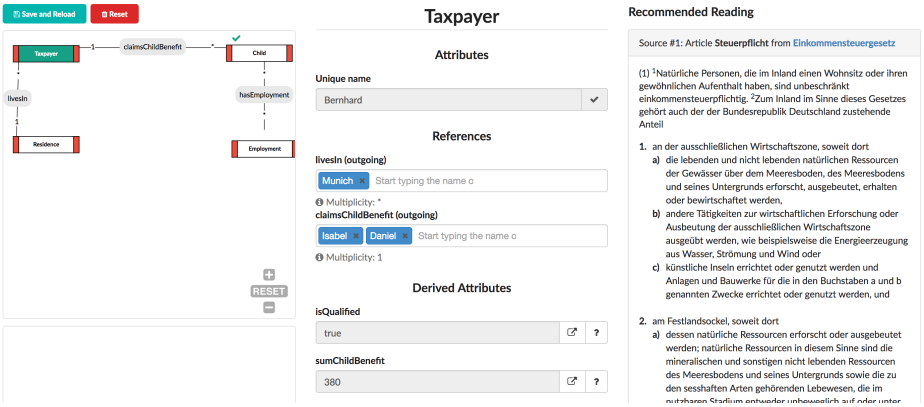

Fig. 4. The front-end of the knowledge acquisition component consisting of three areas: a) model overview (left), b) a form-based input dialog distinguishing between attributes, relations and derived attributes (middle), and c) a set of textual documents to provide additional information about the decision structure (right).

in to three views: the model view on the left, the form view in the middle, and the document view on the right:

Model view The model view is an interactive JavaScript component providing an overview of the semantic model. It enables users to select and unselect types, such as the 'Taxpayer' (see Figure 4), which is highlighted and to which details are shown in the form view and document view. This view shows the relations between types and the cardinalities of those relations. Note, that multiple relations can exist between two types although this is not shown in Figure 4.

Form view The area in the middle of the screen is the form view which is vertically divided into 4 regions, namely the title of the selected type, a list of attributes, a list of relations, and a list of derived attributes. This view offers the user the functionality to graphically add new model instances and provide them with proper facts (attributes and relations). The attributes form the atomic properties that belong to a particular type and the references indicate the relations to instances of another type, e.g., 'Residence' or 'Child'. The derived attributes list shows the evaluated MxL expressions. In addition it offers the functionality to access the explanation dialog by clicking on the question mark beside the input box of a derived attribute (see Section 5.1). Since all attributes are strongly typed, the user interface prevents the input of wrongly typed facts, e.g., number instead of boolean.

Document view The document view allows access to information from a textual resource that was used during the creation of the model. This is considered be valuable since it provides additional information to the form view based on the legislative or jurisdictional texts. Figure 4 shows article 1 entitled "Steuerpflicht" (engl. tax liability) from the the German tax law.

The knowledge acquisition component is a central view of the system and allows the creation of new model instances and the inspection of existing ones. The modeling component is strictly separated from this view. It is not possible to modify the model in the knowledge acquisition phase.

It is necessary to carefully distinguish between the interpretation and application, e.g., subsumption, of a legal norm. Transferring the idea of interpretation of norms to the model based reasoning approach, the interpretation can be considered as the modeling phase. During the modeling phase a legal expert or a group of legal experts formalize the decision structures and specify the required types, attributes, and relations. During a subsequent subsumption phase, legal experts provide the system with facts that constituent themselves in instances of types with concrete attributes and relations. Given this instantiation as input, the model execution component deductively derives new information. It is not necessary that every attribute of an instance is provided the execution component can reason on partially filled types and attributes as well. It is designed to determine as much new information as possible based on the given input.

Once the information of the reasoning structures are modeled and the instances with their attributes and relations are provided, the system is capable of automatically inspecting the reasoning structures regarding data flows and dependencies, which is going to be discussed in the next chapter.

5 Analysis of decision structures for end-users

5.1 Attribute-based dependency trees

Derived attributes, which semantics are specified using the model-based expression language, can potentially consist of complex and nested functional expressions. Those expressions are evaluated before the attribute is accessed in the model store. This read event triggers the evaluation and the execution of the expressions.

A dependency tree for a given derived attribute can automatically be generated by the inference analysis component and forwarded to the dialog component which renders the resulting tree. Thereby, the leaves of the given tree denote either constants, atomic variables, such as 'isEmployed' or 'isJobseeking' (both boolean), or other derived attributes, such as '#age'. The nodes within the tree constitute the various logical and arithmetical operations, such as 'GreaterThan' or 'And', that evaluated during the execution.

The dependency tree view is widely used in semantic program analysis. It enables end-users to understand and reconstruct how the different input parameters contribute to the final result. Consequently, this view fosters the comprehensibility and traceability of a complex decisions structures.

5.2 Interactive data flow graphs

In addition to means for collaborative modeling, model based reasoning, and inspection of attribute-based dependency trees the system offers a holistic perspective to view and explore the data flow throughout a semantic model. This

data flows can again be represented as directed graphs, where the nodes represent attributes or types and the edges represent the usage or contribution within a subsequent type or derived attribute. Thereby, the inference analysis component inspects the data flow during the execution of expressions and passes this information to the dialog component, which renders and visualizes the interactive graphs.

For instance, the data flow graph allows the analysis of contributing variables to the derived attribute 'isQualifiedChild' of a child (Equations 9-12 in Section 3.2). Thereby, it can be seen which attributes influence each other and how it is determined whether a child is eligible for retrieving a benefit or not. It also allows the inspection, which facts are given by a particular instance and how a change in the facts would influence the different attributes resulting into claims that someone has or has not.

6 Conclusion

This paper is a contribution enabling end-users to create executable models representing the semantics of statutory texts, i.e. laws. Based on the theory and prior approaches in the domain of artificial intelligence, in particular legal expert systems and ontologies, we designed an implemented a model based expert system focusing on the end-users perspective. We divided the system into three different components: a model store, a model execution component, and an interaction component.

The model store is a meta-model based information systems persisting the model, i.e. schema, consisting of types, attributes, derived attributes, relations and the facts, i.e. instances of the model. The model execution component is a deductive reasoning engine providing a domain specific language enabling end-users to create executable rules defining the derived attributes. The interaction component is a web based application fostering collaborative access to the model store to create and maintain models and to provide facts. In addition the system has components to analyze the dependency tree of derived attributes and the data flow within a model.

References

1. T. Bench-Capon, M. Araszkiewicz, K. Ashley, K. Atkinson, F. Bex, F. Borges, D. Bourcier, P. Bourgine, J. G. Conrad, E. Francesconi, T. F. Gordon, G. Governatori, J. L. Leidner, D. D. Lewis, R. P. Loui, T. L. McCarty, H. Prakken, F. Schilder, E. Schweighofer, P. Thompson, A. Tyrrell, B. Verheij, D. Walton, and A. Wyner, "A history of AI and Law in 50 papers: 25 years of the international conference on AI and Law," *Artificial Intelligence and Law*, no. 20, pp. 215–319, 2012.
2. G. Sartor, *Legal reasoning: A cognitive approach to the law*, ser. A treatise of legal philosophy and general jurisprudence. Dordrecht: Springer, 2005.
3. E. L. Rissland, K. D. Ashley, and R. P. Loui, "AI and Law: A fruitful synergy," *Artificial Intelligence*, vol. 150, no. 1–2, pp. 1–15, 2003.

4. T. M. van Engers and R. van Doesburg, "First steps towards a formal analysis of law," *Proceedings of eKNOW*, 2015.
5. E. Francesconi, Ed., *Semantic processing of legal texts: Where the language of law meets the law of language*. Springer, 2010.
6. S.-H. Liao, "Expert system methodologies and applications—a decade review from 1995 to 2004," *Expert systems with applications*, vol. 28, no. 1, pp. 93–103, 2005.
7. H. Prakken and G. Sartor, "Law and logic: a review from an argumentation perspective," *Artificial Intelligence*, vol. 227, pp. 214–245, 2015.
8. K. D. Ashley and E. L. Rissland, "A case-based approach to modeling legal expertise," *IEEE expert*, vol. 3, no. 3, pp. 70–77, 1988.
9. M. Aikenhead, "Uses and Abuses of Neural Networks in Law, The," *Santa Clara Computer & High Tech. LJ*, vol. 12, p. 31, 1996.
10. P. Gerathewohl, "Erschließung unbestimmter Rechtsbegriffe mit Hilfe des Computers: Ein Versuch am Beispiel der angemessenen Wartezeit bei § 142 StGB," Dissertation, Eberhard-Karls-Universität, Tübingen, 1987.
11. S. T. Timmer, J.-J. C. Meyer, H. Prakken, S. Renooij, and B. Verheij, "A structure-guided approach to capturing Bayesian reasoning about legal evidence in argumentation," in *Proceedings of the 15th International Conference on Artificial Intelligence and Law*, 2015, pp. 109–118.
12. E. Francesconi, Ed., *Proceedings of LOAIT 2010 -: IV Workshop on Legal Ontologies and Artificial Intelligence Techniques*.
13. A. Wyner, "An ontology in OWL for legal case-based reasoning," *Artificial Intelligence and Law*, vol. 16, no. 4, pp. 361–387, 2008.
14. P. Casanovas, M. A. Biasiotti, E. Francesconi, and M. T. Sagri, Eds., *Proceedings of LOAIT '07: II Workshop on Legal Ontologies and Artificial Intelligence Techniques*, 2007.
15. S. Ramakrishna and A. Paschke, *A Process for Knowledge Transformation and Knowledge Representation of Patent Law*. Cham: Springer International Publishing, 2014.
16. M. B. Islam and G. Governatori, "Ruleoms: a rule-based online management system," in *Proceedings of the 15th International Conference on Artificial Intelligence and Law*. ACM, 2015, pp. 187–191.
17. Object Management Group, "Unified Modeling Language (UML) 2.4.1 Infrastructure." [Online]. Available: http://www.omg.org/spec/UML/2.4.1/
18. ——, "Business Process Model and Notation (BPMN), Version 2.0."
19. ——, "Case Management Model And Notation Version 1.0."
20. ——, "Decision Model and Notation Version 1.0."
21. G. Governatori, Ed., *Law, logic and business processes: Requirements Engineering and Law (RELAW), 2010 Third International Workshop on*, 2010.
22. B. Waltl, F. Matthes, T. Waltl, and T. Grass, "LEXIA: A data science environment for Semantic analysis of german legal texts," *Jusletter IT*, 2016.
23. T. Reschenhofer, I. Monahov, and F. Matthes, "Type-safety in EA model analysis," *IEEE EDOCW*, 2014.
24. T. Jandach, *Juristische Expertensysteme: Methodische Grundlagen ihrer Entwicklung*. Berlin and New York: Springer-Verlag, 1993.