rOWLer - A hybrid rule engine for legal reasoning

Johannes Scharf

University of Vienna, Faculty of Law, Vienna, Austria
johannes.scharf@gmx.at

Abstract. In this paper rOWLer, a hybrid rule engine for legal reasoning is presented. The engine combines the expressiveness of rules and ontologies to enable legal reasoning – hence the name "rOWLer". It is tailored for use in public administration (tax law, pension law, social benefits law, etc.) and provides a flexible architecture, in particular concerning amendments, which allows for adaption to different requirements.

Keywords: Rules, ontologies, OWL, legal reasoning, public administration

1 Introduction

The development of rOWLer is part of the PhD thesis of the author[1] and draws on experiences gained by modelling legal norms with Java and OWL 2. This research tries to fill the gap between the syntactical representation of norms (in XML or other formats) and the need of public administration for a powerful, yet easy to use and customizable legal rule engine. The architecture of rOWLer is aligned with the semantic web stack and is compatible with LegalRuleML [1], an upcoming standard for modelling legal rules. Present software solutions could be improved, following the theoretical models available.

1.1 Motivation

The use of logic-based knowledge systems[2] in public administration (e.g. in tax law) dates back to the 1970s in Austria, but there is still no standard or unified methodology for implementation available. Formalization of statutes in practice happens mainly in an ad hoc fashion by the software expert often without considering legal theory at all.

Although the current models of law are rather useful and accepted in practice they have several severe drawbacks. For instance they violate the isomorphism principle in

[1] Johannes Scharf works as a software engineer at the federal computing center (Bundesrechenzentrum) in Vienna and a PhD researcher at the University of Vienna.

[2] These systems are mostly "production systems" formalizing law by using thousands of if-then-else statements.

a dynamic legal environment which makes maintenance a daunting task. Moreover the legal dynamics (change of law over time) caused e.g. by amendments cannot be handled appropriately. Usually a kind of monotonic reasoning is used which "simulates" defeasible reasoning to some extent. However this approach is very limited in use and can only capture a few aspects of legal reasoning.

The author's PhD thesis tackles these challenges and claims that legal theory and approaches from AI and Law can improve computable models of law used in practice today. In the long run a flexible framework for building legal expert systems is needed which builds on open standards and implements best practices to foster reuse. Such a framework would also need to be complemented by a unified methodology for formalizing legal norms.

The contribution of this research towards a common framework is the development of a solid temporal model which is capable of handling legal change in an efficient manner, e.g. determining applicable rules according to the temporal relations of the case. This supports the development of clean and well-structured models of law and thus decreases maintenance costs. The technical architecture of rOWLer follows a modular approach adhering to best practices[3] from software engineering and can be perceived as an extensible framework for building legal expert systems. This complements efforts to acquire an acknowledged standard for the rule layer of the semantic web cake.

2 Architecture

The architecture of rOWLer consists of three main layers complemented by an electronic document repository, namely the process layer, the rule layer and the ontological layer. What follows is a short overview of the architectural layers of rOWLer, each providing a different view on law and legal rules.

- Process Layer: The process layer formalizes the legal procedure and is responsible to handle the dialog between the applicant and the public agency. It collects the relevant facts by automatic and manual means and interacts with the rule layer to continuously provide preliminary results until the final decision. The authorizing person is asked by the system for decision if a "hard" rule should be applied.
- Rule Layer: This layer contains the formal rules and the inference engine. It drives legal reasoning by retrieving necessary information like facts from the ontology and providing results to the process layer above.
- Ontological Layer: The ontological layer supports the layers above by shallow reasoning on the knowledge base staying within OWL 2, preparing it for more complex reasoning using rules. Especially by data completion, reasoning on mate-

[3] This ensures more clean and maintainable code which is at the same time easier to understand and read.

rial circumstances (claims, facts and proofs) and legal concepts by deriving inferences.

- Electronic Document Repository: This layer complements the formal model by providing access to electronic documents in Akoma Ntoso [13]. Entities of the other layers, this are rules, concepts, etc., can be linked by using IRIs with legal text. This allows for supporting the decision making by the legal expert by providing statutes, commentaries and judgments as well. Moreover it fosters isomorphism of rules by linking them with their legal basis.

3 Reasoning module and algorithm

Technically the algorithm is encapsulated in a module which integrates the reasoner with the rest of the system and also wraps the temporal model. This thin integration layer is also responsible for deriving the parameters from the facts necessary to call the engine, e.g. the significant date. Often it is required to reason over complex situations which span a longer time period[4]. Such scenarios are handled by the reasoning module which interacts with the reasoner to achieve the overall conclusion.

In the following section the proposed algorithm for reasoning is presented, it has to be mentioned that only a rather sketchy overview is given but no complete logical formalization is provided due to space restrictions.

Basically the algorithm is divided into two separate steps to handle temporal and legal reasoning: (1) Determine which rules are applicable to a case at a certain point in time and (2) apply the rules determined in the first step to the case using defeasible reasoning.

The distinction between temporal reasoning and legal reasoning allows for a separate treatment of both problems. In technical terms each of the steps is encapsulated using an interface with an independent implementation. This approach reduces the complexity of the algorithm by separating the whole problem into smaller pieces, independently of each other, while at the same time fostering better integration, maintenance and testing.

4 Temporal model and reasoning

4.1 Theoretical background

There are several possibilities the legislator can adopt to reduce effort and cost of legal change management [10]. Regardless of the methodology followed by the legislator a computable model of law has to deal with changes of sources of law somehow.

---

[4] For example due to the ruling of the Austrian Supreme Court of Justice regarding continuing obligations the time before an amendment has to be judged according to the old rules and afterwards according to the new ones.

For the purposes of the current model we follow the "direct method" of [10] and assume that each change of the sources of law (e.g. by an amendment) leads to a new consolidated version of a statute, containing untouched, modified and new provisions as well. The old version of the statute and its norms enter out of force before the day the new versions enter into force. This approach reduces the complexity of the temporal model.

From a theoretical perspective this may not fully convince as only some provisions are affected by change and thus enter out of force by implicit derogation. However if the legislator enacted an authentic consolidated version of law no such objections exist, even from a theoretical point of view.

To handle change of law two aspects need to be considered: (1) A solid naming convention for statutes and rules and (2) a versioning model which formalizes the temporal dimensions of law.

Due to limited space only the second aspect will be discussed in the next section. It should just be mentioned, that the used naming convention is aligned with FRBR [14] and a simplified version of the HTTP-based syntax for IRIs of Akoma Ntoso [3, 13] compliant with CEN MetaLex [2].

4.2 Versioning model

Temporal dimensions. According to legal theory the temporal model distinguishes the following temporal dimensions of legal norms (cf. [11])[5]:

- Existence: The period in which the norm is part of the legal system, starting with the day of publication (in an official journal), ended by a subsequent normative action.
- Force: When the norm is in force and thus can be applied by the judge in general. In Austria this period usually starts after the day of publication but can be deferred by vacation legis.
- Efficacy: The period in which facts must have occurred in order for the rule to be applicable is called the efficacy period.
- Applicability[6]: This is the period when a legal norm produces the consequences it establishes.

Usually the periods of force coincides with efficacy and applicability of a norm. However it is possible that the effects of a norm start before its force (retroactivity) or continue after the repeal (ultra-activity). For example the tax law of 2008 should be

[5] It has to be noted, that the terms are not always used homogeneously in literature and are used with different meanings. The terms "efficacy" and "applicability" refer to "Bedingungsbereich" and "Rechtsfolgenbereich" respectively in German legal theory [15].

[6] This refers strictly to temporal applicability, the derogation of norms, e.g. by EU law, is tackled in the second reasoning step of the proposed algorithm.

applied to the income earned in 2008 (efficacy), even if a case should be decided after the 31st of December (applicability)[7].

Versioning legal rules. The versioning model used in rOWLer is based on [12] but has been slightly modified an extended to handle not only statutes (documents) but legal rules as well and also to be capable of determining the norms which are applicable to a case at a certain point in time.

The versions of a statute are ordered linearly in so called "versioning chains" by their date of enter into force. When a new version of law is enacted it is added at the end of the chain right after the last version. The model commits itself implicitly that the periods of force of two distinct provisions never overlap. This ensures the soundness of the linear ordering and the versioning chains.

It is assumed that the time when the changes are applied to the legal text coincides with the time of enter into force of the amended provision. Moreover the publication date of the amended provision is assumed to be the same as the amending provision and is also used as the official version date of the act.

Retroactive modifications. Following [12] to handle retroactive modifications the timeline has to be split virtually in the past creating a new legal situation which has not existed originally in this instant in time. To avoid major change of the temporal model in case of retroactive modifications, the proposed solution is enhanced and adopted to avoid splitting of versioning chains.

Each versioning chain is identified by the publication date of the retroactive modification, which is the date from that the chain is valid and hence points at the "current" legal situation. When a retroactive modification arrives, the current chain gets duplicated and the new chain contains the modified provisions starting after the retroactive change is applied.

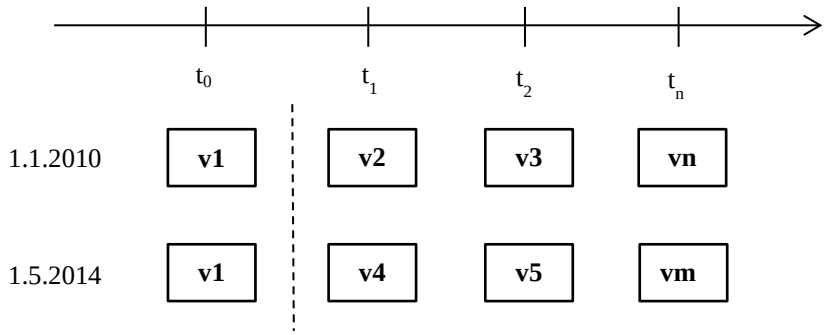

Fig. 1. Example of retroactive modification (adapted from [12])

[7] For the example we assume that the fiscal year coincides with the calendar year.

Fig. 1 shows an example of an amendment published on 5/1/2014 which retroactively modifies v2 at time t_1 and thus leads to a new versioning chain which contains the untouched v1 followed by the amended versions. Virtually the timeline gets split after v1 which is not affected by the modification, as indicated by the dashed line. There is no need to touch the existing chains. The retroactive change of v2 subsequently leads to an adaption of the following versions as well, thus we get the situation described above.

The versioning chains enable the reasoning engine to query the legal situation before and on (or after) the 1st of May 2014 when the retroactive amendment has been published and became part of the legal system. Further it is possible to refer to the "current" legal situation by assigning a variable to the last chain.

When the current chain needs to be put out of service due a retroactive modification the variable "current" simply refers to the new chain, without affecting the rest of the model. Unlike [12] there is no need to split existing chains in case of retroactive modifications.

4.3 Selecting applicable rules

Based on the versioning model and the reflections made in the previous sections a temporal reasoning engine has been designed which is able to compute the legal rules applicable to a case in a given time. In this step the algorithm deals with the "external" time of norms, which guide the lifecycle of the provision and not the "internal" time which is expressed in the rule itself, e.g. when it is obligatory to use winter tires.

To figure out which rules are applicable to a case the engine needs to take the periods of efficacy and applicability into consideration. Accordingly the temporal model needs to be queried with two dates: (1) The view point of the legal system and (2) the "significant" date of the case used to determine the applicable rules. The latter usually depends on the content and type of law (procedural or substantive law). For instance in criminal law the date when the crime has been committed is significant and hence determines which version of law is applicable to the case.

The algorithm adopted by the temporal reasoning engine of rOWLer implements five steps and adheres to non-monotonic reasoning. Hence the temporal engine is able to handle suspending provisions as well which block the effects of a norm temporarily. The rest of this section will give a shallow presentation of each step for better understanding.

1. Query existing norms: The first step builds a snapshot of the legal system at view date, which represents the instant in time when the judge has to apply the rules to the case. In technical terms the last versioning chain with a publication date smaller or equal than the view date is selected.
2. Calculate temporal dimensions: The temporal intervals of the norms selected in the first step are fully calculated by using the norm's static timestamps (startForce, endForce, etc.).

3. Find applicable rules: The third step tries to figure out if a norm should be applied to a case by investigating its dimensions efficacy and applicability. A norm is basically applicable if the view date[8] is captured by the interval of applicability and the significant date of the case occurs within the interval of efficacy of the rule. Formally this can be expressed as follows: $(t_{startEfficacy} \leq significantDate \leq t_{endEfficacy}) \wedge (t_{startApplicability} \leq viewDate \leq t_{endApplicability})$.
4. Suspension of norms: Sometimes the effect of legal rules is suspended by other norms, to give society the chance to adapt their behavior according to the new provisions. The engine needs to handle such suspending norms in a non-monotonic fashion removing the basically applicable rules[9].
5. Resolving references: Finally the engine needs to resolve static and dynamic references. From the conceptual point of view references are normative conditionals whose consequences are not deontic but technically include other rules into the current statute. After including the referenced provisions, the algorithm has to be applied to them in a recursive way. The algorithm stops if no more references need to be resolved.

Basically a simple implementation of the algorithm above could stop after the 3rd step to handle many cases. However, a sophisticated implementation may need to handle suspension of norms and the resolution of references as well.

5 Modelling norms

Following Kelsen [6] we assume in accordance with legal theory that norms have basically the following structure: If $A_1,\ldots,A_2$ then B; where "$A_1,\ldots,A_2$" are the conditions of the norm, "B" is the legal effect and "if…then" is a normative conditional. Norms are therefore formalized using rule objects[10] consisting of antecedent and consequent. Technically rules are represented by an interface called `Rule`. With this abstraction in place it is possible to represent the basic deontic notions, including permission, obligation and prohibition, but more complex Hohfeldian concepts as well.

5.1 Presenting rule priorities

In law we have to deal with implicit (lex specialis, lex posterior) and explicit exceptions between norms. A computable model of law must be able to represent both kinds of exceptions to reflect the way statues are usually written, organized in general rules and exceptions.

---

[8] We assume that the "view date" of the legal system coincides with the point in time when the judge has to apply the rules to the case.

[9] Therefore temporal reasoning adheres to a kind of defeasible reasoning too.

[10] We use an object-oriented model.

In AI and Law different methods to solve conflicts between rules have been proposed, namely *specificity*, *weight* (salience) and *preference relation*. The model of rOWLer supports weights and preference relations by using interfaces `WeightedRule` and `PreferenceRelation` respectively.

Conflicts between rules are resolved by ordering rules using an implementation of `RuleOrderingStrategy`. The strategy inspects all rules to order the rules supporting all of the methods above, using explicit and implicit information as well. The rules are placed in a network representing their ranking and wrapped by a dynamic proxy[11] at runtime implementing `SuperiorityRelation`.

A `SuperiorityRelation` represents an abstract concept describing the binary relationship between two rules[12], covering specificity, weight and preference relation as well. This abstraction allows for a dynamic creation of arbitrary relations between rules, e.g. of *lex superior* and *lex inferior* by inspecting the law making institutions modelled in the ontology and linked with the rules.

The model enhanced with superiority relations between rules builds the foundation for qualifying the rules as defeater, defeasible and strict in the sense of defeasible logic [9]. Further it enables the use of a defeasible engine like SPINdle [7] for reasoning or the implementation of a custom engine built on an algorithm like [8].

Due to severe space limitations it is impossible to provide more details and to show how deontic rules, metadata and isomorphism are handled by the formal model.

6 Related work

JBoss Drools[13] is an open-source business rule engine and as such uses production rules as data structure. Since version 6 it is based on "PHREAK" a monotonic algorithm supporting forward and backward chaining.

Although drools performs well with thousands of rules and has a nice declarative style for writing rules, it is not suited for the legal domain. First of all it only supports monotonic reasoning and thus cannot deal with incomplete data. Second the time model of Drools does not support the temporal dimensions of law and thus would have to be extended to handle legal change over time. Compared to Drools, rOWLer adheres to defeasible reasoning and its temporal model is well suited for the legal domain.

---

[11] The architecture of rOWLer is consistently based on interfaces which allows for using Java's dynamic proxying facilities.

[12] In this a sense superiority relation resembles a preference relation but in contrast to the latter it is an abstraction whose instances are built dynamically at runtime by the engine.

[13] By referring to "Drools" we mean "Drools Expert" which is the rule engine of the Drools platform.

SPINdle [7] is another open-source rule engine which supports defeasible logic and modal defeasible logic as well. Unlike Drools, which is based on a monotonic algorithm, it is capable of defeasible reasoning over theories with thousands of rules. SPINdle gives basic support for time and intervals but cannot handle the temporal dimensions (force, efficacy, applicability) of legal norms. rOWLer is built on a sophisticated versioning model supporting temporal reasoning to determine applicable provisions.

The rules in SPINDle are heavy based on literals. Basically the conclusion of a rule is a literal or its negation. To formalize norms we need a representation of a rule which allows for representing richer conclusions, e.g. a calculation or the inclusion of other norms in case of references. Therefore rOWLer supports a richer object model supporting different kind of rules which are executed by using an appropriate strategy.

However, it would be nice if SPINdle could be used as defeasible rule engine embedded inside rOWLer. rOWLer's modular architecture and algorithm would allow for such an integration.

7 Conclusions and future work

Compared to present approaches in public administration, rOWLer is aligned with legal theory and fosters defeasible reasoning, while maintaining isomorphism with the sources of law. To cope with legal change over time a solid temporal model has been developed with formalizes the temporal dimensions of law and further is able to decide which norms should be applied to a case at a certain point in time. By using a viewpoint the model is also capable of handling retroactive modification by providing the historic and current version of a statute after the amendment. Present implementations used in practice lack a sophisticated temporal model for handling legal change which increases code complexity and leads to severe maintenance problems.

At the moment rOWLer is designed as a single-agent system and the reasoning engine is optimized to deal with statutes with a rather mathematical content like tax law or "easy" cases[14] in the terminology of Hart. However, the model of rOWLer is flexible enough to be extended in the future to handle "hard" cased as well, e.g. by providing the legal expert with different alternatives for decision making and integrating more sophisticated argumentation systems like Carneades [4].

In the future the conceptual model needs to be refined, especially with regard to the representation of norms and defeasible reasoning. Feasibility of the theoretical approach should be evaluated by developing a prototype in Java, which has become the "mainstream" programming language nowadays.

---

[14] "Easy" cases can be largely decided "mechanically" by deducing the required result from the rule and the facts. "Hard" cases are ones for Hart in which the facts fall within the "penumbra" of the meaning of the words in the applicable rule. These cases require the judge to exercise discretion [5].

Acknowledgements. I would like to thank my supervisor Erich Schweighofer of the Faculty of Law, University of Vienna, Centre for Computers and Law, for help and guidance through this challenging research.

References

1. Athan, T. et al.: OASIS LegalRuleML. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law - ICAIL '13. pp. 3–12 ACM Press, New York, USA (2013).
2. Boer, A., van Engers, T.: A MetaLex and Metadata Primer: Concepts, Use, and Implementation. In: Sartor, G. et al. (eds.) Legislative XML for the Semantic Web. pp. 131–149 Springer Netherlands, Dordrecht (2011).
3. Francesconi, E.: Naming Legislative Resources. In: Sartor, G. et al. (eds.) Legislative XML for the Semantic Web. pp. 49–74 Springer Netherlands, Dordrecht (2011).
4. Gordon, T.F.: An Overview of the Carneades Argumentation Support System. In: Reed, C. and Tindale, C.W. (eds.) Dialectics, Dialogue and Argumentation. An Examination of Douglas Walton's Theories of Reasoning. pp. 145–156 College Publications, London (2010).
5. Hart, H.L.A.: The Concept of Law. Clarendon Press, Oxford (1994).
6. Kelsen, H.: Allgemeine Theorie der Normen. Manz, Wien (1979).
7. Lam, H.-P., Governatori, G.: The Making of SPINdle. In: Governatori, G. et al. (eds.) Rule Interchange and Applications. pp. 315–322 Springer, New York (2009).
8. Maher, M.J.: Propositional defeasible logic has linear complexity. Theory Pract. Log. Program. 1, 06, 691–711 (2004).
9. Nute, D.: Defeasible Logic. In: Bartenstein, O. et al. (eds.) Web Knowledge Management and Decision Support. pp. 151–169 Springer Berlin Heidelberg, Berlin (2003).
10. Palmirani, M.: Legislative Change Management with Akoma-Ntoso. In: Sartor, G. et al. (eds.) Legislative XML for the Semantic Web. pp. 101–130 Springer Netherlands, Dordrecht (2011).
11. Palmirani, M. et al.: Modelling temporal legal rules. Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11. pp. 131–135 ACM Press, New York, New York, USA (2011).
12. Palmirani, M., Brighi, R.: Time Model for Managing the Dynamic of Normative System. In: Wimmer, M.A. et al. (eds.) Electronic Government. pp. 207–218 Springer Berlin Heidelberg, Berlin Heidelberg (2006).
13. Palmirani, M., Vitali, F.: Akoma-Ntoso for Legal Documents. In: Sartor, G. et al. (eds.) Legislative XML for the Semantic Web. pp. 75–100 Springer Netherlands, Dordrecht (2011).
14. Saur, K.G.: Functional Requirements for Bibliographic Records: Final report. IFLA Study Group on the Functional Requirements for Bibliographic Records, München (2009).
15. Walter, R. et al.: Grundriss des österreichischen Bundesverfassungsrechts. Manzsche Verlags- und Universitätsbuchhandlung, Wien (2007).