Judging Amy: Automated Legal Assessment using OWL 2

Saskia van de Ven, Rinke Hoekstra, Joost Breuker,
Lars Wortel, and Abdallah El-Ali

Leibniz Center for Law, University of Amsterdam,
s.vandeven@uva.nl, hoekstra@uva.nl, breuker@uva.nl,
l.l.wortel@uva.nl, aali@science.uva.nl

Abstract. One of the most salient tasks in law is legal assessment, and concerns the problem of determining whether some case is allowed or disallowed given an appropriate body of legal norms. In this paper we describe a system and Protégé 4 plugin, called OWL Judge, that uses standard OWL 2 DL reasoning for legal assessment. Norms are represented in terms of the LKIF Core ontology, as generic situation descriptions in which something (state, action) is deemed obliged, prohibited or permitted. We demonstrate the design patterns for defining the norms and actual cases. Furthermore we show how a DL classifier can be used to assess individual cases and automatically generate a lex specialis exception structure using OWL Judge. We illustrate our approach with a worked-out example of university library regulations.

Introduction

One of the most salient tasks in law is legal assessment, and concerns the problem of determining whether some case is allowed or disallowed given an appropriate body of legal norms. For legal experts this process becomes more and more difficult as the amount of legislation continuously grows and the complexity of legislative documents increases. Moreover, the structure of legal documents differs from those in other domains. Legislation tends to be highly interconnected – and interdependent – and normative conflicts between different regulations are resolved by meta legal rules, such as *lex specialis* and *lex posterior*. The application of these rules can result in complex exception structures, which can be difficult to keep track of for the average citizen. [12] presented a way to resolve *lex posterior* (the more recent norm takes precedence) conflicts between concept definitions. Conversely, this paper is focused on the resolution of *lex specialis* exceptions between norms, the legal principle that states that a more specific norm takes precedence over a more general one.

Legal knowledge-based systems allow us to manage the complexity of legislation and enable online e-government services. Transactions that require the application of complex legislation, such as tax benefits or social security administration, are increasingly processed online. This gives citizens and businesses more direct access to a personalised and transparent insight in their rights and

duties. Although there already exist several such systems, the wider acceptance and dissemination of legal knowledge technology is hampered by the lack of an open platform.

The Legal Knowledge Interchange Format (LKIF), developed within the ESTRELLA[1] project, aims to provide an open platform for legal knowledge exchange between e-government services [1,10]. HARNESS[2] is the working title of a knowledge-based system that is currently being developed within this project. Its architecture is based on the separation of control knowledge, such as problem solving methods, from a domain theory [16,3]. The general architecture allows for the implementation of multiple tasks, such as legal planning and argumentation, though it currently only supports legal *assessment*. This task is solely implemented using the OWL 2 DL language.

In the following sections we describe the requirements for legal assessment and explain our approach. We present OWL Judge which is developed to assist users in the task of assessing their individual case. All of this will be illustrated with a worked-out example of university library regulations.

1 Legal Assessment

Clancey [4] identified different types of rules involved in the medical domain theory of the MYCIN system: identification, causal, world fact and domain fact rules. Similarly, the domain theory of *legal* knowledge based systems is not homogeneous either. [15,17] give a general breakdown of the types of knowledge and their dependencies involved in the legal domain. Valente's Functional Ontology of Law (FOLaw) describes the legal system as an instrument to influence society and reach certain social goals, i.e. it exists to fulfil a *function*. The legal system can be viewed as a "social device operating within society and on society, and whose main function is to regulate social behaviour" [15, p. 49].

FOLaw distinguishes seven types of knowledge: commonsense knowledge, world knowledge, normative knowledge, responsibility knowledge, meta-legal knowledge, creative knowledge and reactive knowledge. Any legal knowledge based system will incorporate at least one of these knowledge types.

The most characteristic category of legal knowledge is *normative knowledge*, which reflects the regulatory nature of law. It has two functions: prescribing behaviour, and defining a standard of comparison for social reality. A norm expresses an idealisation: what ought to be the case (or to happen) a ccording to the will of the agent that created the norm [11]. *Meta-legal knowledge* governs relations between norms, and is applied when solving normative conflicts [6], and determining the *preference ordering* of norms [2]. Legal principles such as *lex specialis* are types of meta-legal knowledge. Because law governs the behaviour of agents in the world, it must contain some description of this behaviour. This

[1] The European project for Standardized Transparent Representations in order to Extend Legal Accessibility, see http://www.estrellaproject.org/.

[2] Hybrid Architecture for Reasoning with Norms Exploiting Semantic web Services

world knowledge functions as an epistemological interface between the legal system and social reality. World knowledge is an abstraction of commonsense [15]. This principle lies at the heart of the LKIF Core ontology of basic legal concepts [10,**?**].

Legal assessment concerns the problem of determining whether some case is allowed or disallowed given an appropriate body of legal norms. It involves the expression of an individual case, an interpretation of that case in terms of the world knowledge, and the application of normative knowledge to determine whether the case constitutes a norm violation. More precisely, the following steps are needed to perform this task:

Case Input Create a description of the case that needs to be assessed. The facts that are provided should correspond to terms are used in the norms, i.e. both the norms and the individual case should commit to the same domain ontology.

Match The individual case is matched against the set of norms.

Resolving conflicts Because it is not unlikely that the individual case matches more than a single norm, the system identifies conflicting norms and decides which norm has priority over the other norms (e.g. according to the principle of lex specialis).[3]

Final verdict Provide an outcome of the normative qualification of the case. It is desirable to present an *explanation*, e.g. why a certain norm applies to the given input, showing which terms in a norm match the individual facts of the case. Moreover the answer should be *justified* by reference to the norms and original source. [8].

In the following sections we outline the representation in OWL 2 DL of the three types of knowledge necessary for legal assessment. A domain ontology for terminological knowledge defines the concepts used as the basic building blocks for specifying case descriptions and norms. The ontology is a specialisation of the LKIF Core ontology. The norms are specified in a separate OWL file, and define the regulations that govern the situations expressible using the domain ontology. They are expressed as generic situations in which a state or action is qualified as undesirable, permitted or prescribed (see Figure 1). The individual case description represents the facts that characterise an actual situation and is expressed in terms of the domain ontology as a set of related OWL individuals. Furthermore we will provide the corresponding OWL DL design patterns and show how a description classifier can be used to perform legal assessment of an individual case.

2 Norms and Generic Cases

A norm is represented as a deontic qualification of a generic case [15] (GC), which is a conjunction of conditions that together form a description of the situation

[3] Note that the outcome of this step does not always result in a single match. Several norms may apply that have a conflicting deontic qualifier, but are not each other's exception.

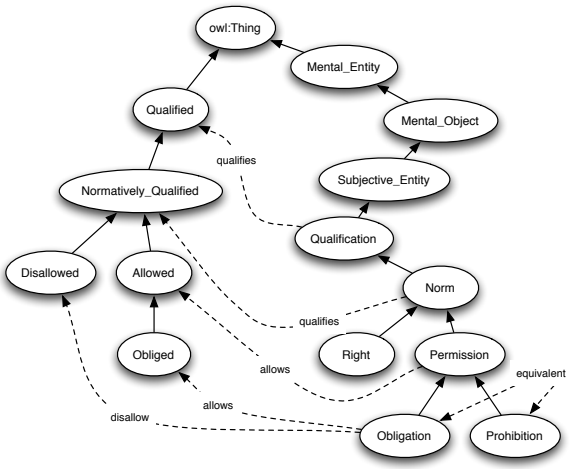

Fig. 1. Qualifications and Norms

expressed by the norm. A GC Σ is defined as a set of conditions $\sigma_1 \sqcap ... \sqcap \sigma_n$ in conjunctive normal form. Conditions are class axioms composed of classes and properties defined by the domain ontology. An individual case C is a set of grounded propositions $\{c_1, ..., c_n\}$, called circumstances, that describe a certain state of affairs. The match between a GC and an individual case is through realisation. For instance, a generic case GC_1 that is used in the definition of the norm "Students are not allowed to rent books":

$$\mathsf{GC}_1 \equiv \mathsf{Student} \ \sqcap \ \exists \mathsf{checks\_out} \ \mathsf{Library\_Book}$$

is trivially satisfied by the individual case:

$$\{Amy : \mathsf{Student}, history\_book : \mathsf{Library\_Book}, Amy \ \mathsf{checks\_out} \ history\_book\}$$

Obviously, a more specific case GC_2 will be subsumed by GC_1. However the above example already shows that just representing a norm by a generic case is not enough. From the generic case itself, it does not become clear whether the expressed situation is allowed or not. Therefore we need the following deontic notions that enable us to normatively qualify a generic case: permission, obligation and prohibition. These notions are part of the LKIF Core ontology which is described in a bit more detail at the end of this section.

As the goal of our system is normative assessment, we would like to detect violations of the *default situation*. Norms apply only to cases that differ from this default situation. The majority of legislative documents by default *allow* a situation, unless some norm(s) state(s) otherwise. In some regulations, the default situation is *disallowed*. This is for instance the case in permit systems. The default situation should therefore be included as a generic case in any normative representation. All other generic cases specified by norms are subsumed by

Notion	Match	Default Allowed	Default Disallowed
Prohibition	**no**	no effect	***possible effect***
	yes	has effect	no effect
Permission	**no**	no effect	has effect
	yes	no effect	has effect
Obligation	**no**	***possible effect***	no effect
	yes	has effect	has effect

Table 1. Behaviour of deontic notions with respect to the default

this default generic case. This allows us to make explicit any exceptions to the default by more specific norms.

The way in which deontic operators interact can be rather complex. We therefore introduce *design patterns* to facilitate the specification of norms. The following paragraphs describe the behaviour of norms compared to the default, and provide the design patterns (see also Table 1).

Default situation: allowed In the "most common" case, everything is by default allowed. A match of an individual case with the GC of a *permission* does not have any normative consequences for example. Similarly, when the individual case does not match a *prohibition*, no normative consequences follow. On the other hand, a match with the GC of a prohibition, constitutes an exception to the default. These types of exceptions should be detected and given precedence over the default situation. However, this simplistic matching process becomes more complex in the case of no match with the GC of an obligation.

We take a fictive example from the library domain again: "Students are obliged to show their library card in order to rent a book. If in this example, someone does want to rent books (a match with the context), but does not show his library card (which is the specific obliged part), the obligation is violated. However there would be no match with the generic case which expresses this (only a partial match), we would have failed to detect a violation of the default. To be able detect these violations, the GC is partitioned into that part which is explicitly *allowed* (A_Σ) and that part which is explicitly *disallowed* (D_Σ) by the obligation. Moreover a norm is typically only addressed towards a certain element of the GC, given some *context*. We therefore distinguish between the context of a norm and that part of the GC which the norm is directed towards. The pattern we follow in the representation of obligations is therefore as follows:

$$
\begin{array}{rll}
 & A_\Sigma & \equiv \sigma_1 \sqcap ... \sqcap \sigma_n \sqcap \sigma_{n+1} \\
 \nearrow & & \equiv \Sigma_1 \sqcap \Sigma_2 \\
O_\Sigma & & \\
 \searrow & & \\
 & D_\Sigma & \equiv \sigma_1 \sqcap ... \sqcap \sigma_n \\
 & & \equiv \Sigma_1
\end{array}
$$

Where the context is defined as Σ_1: $\sigma_1 \sqcap ... \sqcap \sigma_n$ (a student rents a book), and the obliged part of the GC is Σ_2: σ_{n+1} (showing the library card). Renting a book is disallowed, but renting a book and showing a library card is allowed, which is the desired behaviour.

Default situation: disallowed Of course, where the default situation is disallowed, the normative qualifications differ significantly. For instance, if a norm expresses a *permission*, and there is *no* match with an individual case, we are still in line with the default situation. However, a match constitutes a direct exception to the default. Similarly, for obligations and prohibitions the exceptions change: a compliance with the allowed part of an obligation is an exception, where a compliance with the disallowed part of a prohibition is not. Therefore, if there is no match in case of a prohibition: we might have failed to detect a violation of the default. A prohibition expresses a *disallowed* situation (D_Σ), but also implicitly represents an *allowed* (A_Σ) situation (the complement of the disallowed case is an allowed situation). In case of the example "Students that are in the library, are prohibited to make noise" it is allowed to be in the library, but it is disallowed to be in the library and make noise. The design pattern is as follows:

$$
\begin{array}{rll}
 & D_\Sigma & \equiv \sigma_1 \sqcap ... \sqcap \sigma_n \sqcap \sigma_{n+1} \\
 & \nearrow & \equiv \Sigma_1 \sqcap \Sigma_2 \\
F_\Sigma & & \\
 & \searrow & \\
 & A_\Sigma & \equiv \sigma_1 \sqcap ... \sqcap \sigma_n \\
 & & \equiv \Sigma_1
\end{array}
$$

Where the context is defined as Σ_1: $\sigma_1 \sqcap ... \sqcap \sigma_n$ (being in the library), and the prohibited part of the GC is Σ_2: σ_{n+1} (making noise).

The LKIF Core ontology defines the class Norm and its subclasses Permission, Obligation and Prohibition as follows [10,2]:

$$
\begin{aligned}
\mathsf{Norm} &\sqsubseteq \mathsf{Qualification} \sqcap \exists \mathsf{qualifies}\ \mathsf{Normatively\_Qualified} \\
\mathsf{Permission} &\sqsubseteq \mathsf{Norm} \\
&\equiv \exists \mathsf{allows}\ \mathsf{Allowed} \sqcap \forall \mathsf{allows}\ \mathsf{Allowed} \\
\mathsf{Obligation} &\sqsubseteq \mathsf{Permission} \\
&\equiv \exists \mathsf{allows}\ \mathsf{Obliged} \sqcap \exists \mathsf{disallows}\ \mathsf{Disallowed} \\
&\quad \sqcap \forall \mathsf{allows}\ \mathsf{Allowed} \sqcap \exists \mathsf{disallows}\ \mathsf{Disallowed} \\
\mathsf{Prohibition} &\equiv \mathsf{Obligation}
\end{aligned}
$$

The Prohibition and Obligation are equivalent because they are simply two different ways to put the same thing into words: a prohibition to smoke is an obligation not to smoke. The Permission is different in that it allows something, but does not prohibit anything. For the purpose of clarity we will refer to the subclasses of Normatively_Qualified as generic cases. The properties allows and

disallows are disjoint subproperties of qualifies; a norm cannot simultaneously allow and disallow the same situation. Although a GC describes an entire situation, it can be represented as a single class description because the entities in a single situation should be connected. On the other hand, this means the norm's qualification is biased to the class that forms the entry point for describing the GC.

3 The University Library Regulations

Now that we have explained our approach we will illustrate it by means of a worked-out example of a toy domain: university library regulations. We list relevant regulations and introduce a use case. We then show how the regulations are represented, and introduce OWL Judge.
Consider the following university library regulations:

1a) Students registered at this university are allowed to check out a book from this library
1b) Students registered at other universities are allowed to check out a book from this library provided that they are enrolled in at least one course given at this university.
1c) Students who have checked out more than five books are not allowed to check out another book.

One of our students is Amy. She is registered at the University of Amsterdam and has an inquisitive mind: she has checked out 6 books on some general topic:

$$\{\mathit{Amy} : \mathsf{Registered\_Student}, \mathit{general\_book\_1} : \mathsf{Library\_Book}, \mathit{general\_book\_2} : \mathsf{Library\_Book}, \mathit{general\_book\_3} : \mathsf{Library\_Book}, \mathit{general\_book\_4} : \mathsf{Library\_Book}, \mathit{general\_book\_5} : \mathsf{Library\_Book}, \mathit{general\_book\_6} : \mathsf{Library\_Book}, \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_1}, \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_2}, \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_3}\ \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_4}, \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_5}, \mathit{Amy}\ \mathsf{checks\_out}\ \mathit{general\_book\_6}\}$$

According to the regulations, Amy's situation is allowed by article *1a*, but disallowed by article *1c*. However, these two articles contain an intuitive hierarchic pattern: the case described by article *1a* subsumes that of article *1c*. In fact, article *1c* is an exception to *1a*, because *1a* implicitly expresses that students registered at *other* universities are not allowed to check out any books. The expected verdict of the system should therefore be that Amy's situation is *disallowed*, as article *1c* is more specific than article *1a*, the lex specialis principle states that *1c* trumps *1a*.

Although the two relevant articles are article *1a* and *1c*, we first need to specify the default situation. In the library regulations the default situation is

disallowed; students are generally not allowed to check out books, unless stated differently in the norms. The default norm and generic case apply to all situations where a book is checked out:

$$\begin{aligned} \mathsf{Default\_GC} &\sqsubseteq \mathsf{Generic\_Case} \sqcap \exists \mathsf{disallowed\_by}\ \{\mathit{defaultnorm}\} \\ &\equiv \exists \mathsf{checks\_out}\ \mathsf{Library\_Book} \\ \mathsf{Default\_Norm} &\sqsubseteq \mathsf{Prohibition} \sqcap \forall \mathsf{disallows}\ \mathsf{Default\_GC} \\ &\equiv \{\mathit{defaultnorm}\} \end{aligned}$$

Article *1a* states that students registered at this university are allowed to check out a book from this library:

$$\begin{aligned} \mathsf{Art1a\_GC} &\sqsubseteq \mathsf{Generic\_Case} \sqcap \exists \mathsf{allowed\_by}\ \{\mathit{art1a}\} \\ &\equiv \mathsf{Registered\_Student} \sqcap \exists \mathsf{checks\_out}\ \mathsf{Library\_Book} \\ \mathsf{Art1a\_Permission} &\sqsubseteq \mathsf{Permission} \sqcap \forall \mathsf{allows}\ \mathsf{Art1a\_GC} \\ &\equiv \{\mathit{art1a}\} \end{aligned}$$

Article *1c* states that students who have checked out more than five books are not allowed to check out another book:

$$\begin{aligned} \mathsf{Art1c\_GC\_F} &\sqsubseteq \mathsf{Generic\_Case} \sqcap \exists \mathsf{disallowed\_by}\ \{\mathit{art1c}\} \\ &\equiv \mathsf{Student} \sqcap \geq\ 6\ \mathsf{checks\_out}\ \mathsf{Library\_Book} \end{aligned}$$

$$\begin{aligned} \mathsf{Art1c\_GC\_P} &\sqsubseteq \mathsf{Generic\_Case} \sqcap \exists \mathsf{allowed\_by}\ \{\mathit{art1c}\} \\ &\equiv \mathsf{Student} \sqcap \exists \mathsf{checks\_out}\ \mathsf{Library\_Book} \\ &\quad \sqcap \leq\ 5\ \mathsf{checks\_out}\ \mathsf{Library\_Book} \end{aligned}$$

$$\begin{aligned} \mathsf{Art1c\_Prohibition} &\sqsubseteq \mathsf{Prohibition} \sqcap \forall \mathsf{disallows}\ \mathsf{Art1c\_GC\_F} \\ &\quad \sqcap \forall \mathsf{allows}\ \mathsf{Art1c\_GC\_P} \\ &\equiv \{\mathit{art1c}\} \end{aligned}$$

Article *1c* expresses a prohibition where the default situation is disallowed. The GC is therefore partitioned into a part that is disallowed (the prohibition) and the thing that is allowed (the context). In this case it is permitted to check out books, checking out more than 5 books is prohibited. By using a cardinality restriction, checking out less than or exactly 5 books is allowed by this rule.

3.1 Exception structure

As mentioned in the preceding paragraphs, multiple interacting norms can create complex exception structures. As our norms are represented as OWL classes, a DL classifier can generate the subsumption hierarchy of generic cases. This hierarchy actually reflects the lex specialis exception relations between norms. Moreover we can apply the following role inclusion axioms to make these exception

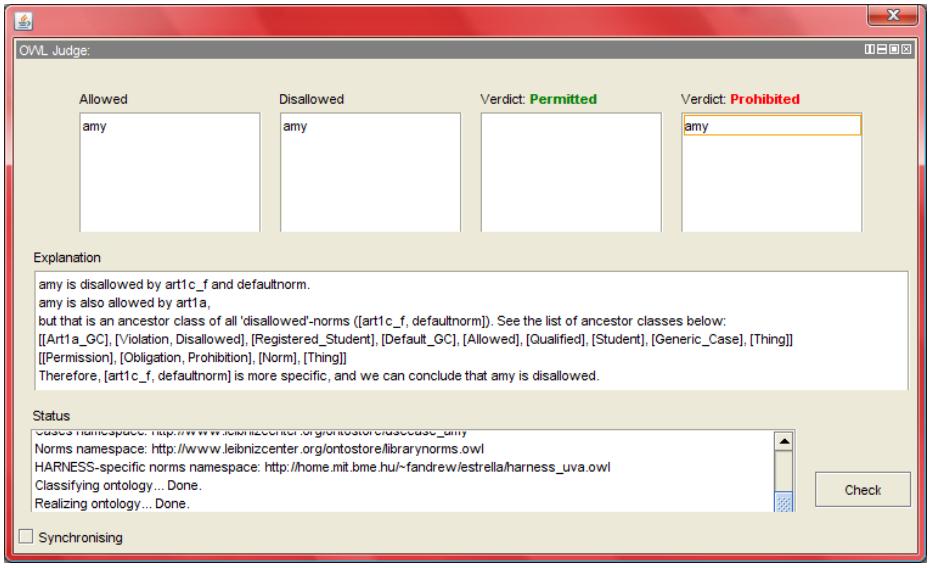

Fig. 2. OWL judge provides a verdict for Amy's case.

relations explicit:

$$\mathsf{allows} \circ \mathsf{disallowed\_by} \subseteq \mathsf{exception}$$
$$\mathsf{disallows} \circ \mathsf{allowed\_by} \subseteq \mathsf{exception}$$

Because norms are nominals that consist of a single individual, these exception relations can be derived automatically, using Pellet for all norms. Take for example the Art1c_Prohibition which disallows Art1a_GC. The individual *art1c*, which represents the prohibition from article *1c*, disallows our individual *amy*, since her situation matches with Art1c_GC. However, *amy* is also classified as Art1a_GC, and therefore *amy* is also allowed_by *art1a*. In this way we can derive the exception relation between *art1c* and *art1a*:

$$\{art1c \ \mathsf{exception} \ art1a\}$$

This inferred exception hierarchy is used by OWL Judge to resolve possible conflicts, and helps to present a final verdict to the user.

3.2 OWL Judge

OWL Judge assists users in the task of legal assessment. It prompts the user to select an OWL file containing the norms, and a file containing a case description. Once the "check" button is pressed, Pellet is called to perform classification and realisation on the given files (see Figure 2).

First OWL Judge checks all individuals that exist in the file containing the case description. Depending on their classification as Allowed or Disallowed, they are displayed in respectively the "Allowed" and/or "Disallowed" column, shown in the top left part of the screen. A user can click on the individuals in those columns and retrieve an explanation of the classification. For the explanation, OWL Judge uses the standard explanation facilities of Pellet [8].

OWL Judge also generates a final verdict, which is especially interesting in situations where multiple norms apply to the individual case (if only one norm applies it is clearly the final verdict as well). If an individual is both Allowed and Disallowed, OWL Judge tries to resolve this conflict by checking whether one of the norms allowing or disallowing it contains an exception relationship with the other norm: a norm trumps another norm, if the generic case of the first norm is more specific. This means some additional querying of the TBox and ABox is done by the plugin.

The most specific one will be displayed in the appropriate "Verdict" column, shown in the top right of the screen. The user can again select the individual to view an explanation of OWL Judge's verdict. The screen shows that the individual *amy* is disallowed by *defaultnorm* and *art1c*, but allowed by *art1a*. The explanation indicates that article 1c is the most specific norm, which makes the final verdict "prohibited".

4 Conclusions and future work

Currently, the individual case description has to be entered using a generic OWL editor, such as Protégé[4] or Topbraid Composer[5]. This of course asks for quite some experience in working with these kind of tools. Moreover a person that creates the case description should have knowledge of what is present in the ontology and in the norms, as the terms in the case description should correspond to either the terms used in the ontology or in the norms. It would be very beneficial to provide a more friendly user interface. Such an interface should for example guide a user in what terms to use and try to complete the case description as much as possible. Some work, cf. [5], suggests that the language used in regulations is highly structured, which suggests a possibility to support a structured language interface along the lines of [14].

OWL Judge as it exists now implements our basic requirements, but there are still some things we would like to address. For instance, it might be interesting to enhance the *explanation* by allowing users to click on a certain individual case to get more information about it. Moreover OWL Judge is not yet capable of determining whether an individual case is *Obliged*, which is also something that needs to be dealt with. Furthermore, we would like to further enhance the explanations with forms of *justification*, such as a reference to the original source of law. Preferably the original source of law is stored in the MetaLex/CEN format

[4] http://protege.stanford.edu
[5] http://www.topbraidcomposer.com

of [19], an open XML interchange format for legal and legislative resources[6]. This enables a direct connection between the original legal source to (specific parts of) our model. This would also form a nice way to interact with the MetaVex editor, a WYSIWYG editor used to create and edit documents in the MetaLex/CEN format [18].

The major advantage of using only OWL 2 DL to represent both conceptual and normative knowledge is that we can use an existing reasoner, Pellet, to perform all reasoning. There is no need to concern ourselves with an interaction between different formalisms. The question is whether OWL 2 DL is expressive enough to suit our needs. In some cases we would like to talk about things at the level of individuals, instead of at the level of concepts. For example, if we want to represent "a student that checks out a book belonging to a course *he* is enrolled in", this is currently impossible to express in DL because of the tree model property[7].

Rule formalisms can easily represent this example just by using variables each time we want to point to a certain student or course. Alternative approaches, such as structured objects ([13]) are very promising as well, but are currently not supported by mainstream reasoners. However, the necessary effect can sometimes be achieved by approximation [9]. Also, others are working to achieve higher expressiveness by using SWRL conditions [7]. Thus far we have found solutions that work for limited cases but cannot be applied in general. Nonetheless, the approach explained in this paper might be sufficient for some domains, and we intend to thoroughly test and evaluate our approach on existing regulations.

References

1. Alexander Boer, Thomas F. Gordon, Kasper van den Berg, Marcello Di Bello, András Förhécz, and Réka Vas. Specification of the legal knowledge interchange format. Deliverable 1.1, Estrella, 2007.
2. Alexander Boer, Tom van Engers, and Radboud Winkels. Mixing legal and non-legal norms. In M-F. Moens and P. Spyns, editors, *Jurix 2005: The Eighteenth Annual Conference.*, Legal Knowledge and Information Systems, pages 25–36, Amsterdam, 2005. IOS Press.
3. Joost Breuker. Components of problem solving. In Luc Steels, Guus Schreiber, and Walter van de Velde, editors, *A Future for Knowledge Acquisition: proceedings of the EKAW-94, European Knowledge Acquisition Workshop*, pages 118 – 136, Berlin, 1994. Springer Verlag.
4. William J. Clancey. The epistemology of a rule-based expert system - a framework for explanation. *Artificial Intelligence*, 20(3):215–251, 1983. First published as Stanford Technical Report, November 1981.
5. E. de Maat, R. Winkels, and T. van Engers. Making Sense of Legal Texts. In G. Grewendorf and M. Rathert, editors, *Formal Linguistics and Law*, Trends in Linguistics - Studies and Monographs (TiLSM). Mouton, De Gruyter, Berlin, (in press) 2008.

[6] http://www.metalex.eu/

[7] See [9] for an in-depth discussion on the problem of identity of individuals.

6. Abdullatif A.O. Elhag, Joost A. Breuker, and Bob W. Brouwer. On the formal analysis of normative conflicts. In H.Jaap van den Herik et al., editor, *JURIX 1999: The Twelfth Annual Conference*, Frontiers in Artificial Intelligence and Applications, pages 35–46, Nijmegen, 1999. GNI.
7. A. Forhecz and G. Strausz. First-order owl conjunctions. In *Proceedings of OWLED 2008 EU*, 2008. Under Review.
8. Christian Halaschek-Wiener, Yarden Katz, and Bijan Parsia. Belief base revision for expressive description logics. In *OWL: Experiences and Directions 2006*, 2006.
9. Rinke Hoekstra and Joost Breuker. Polishing diamonds in OWL 2. In Aldo Gangemi and Jérôme Euzenat, editors, *Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)*, LNAI/LNCS. Springer Verlag, October 2008. To be published.
10. Rinke Hoekstra, Joost Breuker, Marcello Di Bello, and Alexander Boer. The LKIF Core ontology of basic legal concepts. In Pompeu Casanovas, Maria Angela Biasiotti, Enrico Francesconi, and Maria Teresa Sagri, editors, *Proceedings of the Workshop on Legal Ontologies and Artificial Intelligence Techniques (LOAIT 2007)*, June 2007.
11. Hans Kelsen. *General Theory of Norms*. Clarendon Press, Oxford, 1991.
12. Szymon Klarman, Rinke Hoekstra, and Marc Bron. Versions and applicability of concept definitions in legal ontologies. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of OWL: Experiences and Directions (OWLED 2008 DC)*, Washington, DC (metro), April 2008.
13. Boris Motik, Bernardo Cuenca Grau, and Ulrike Sattler. Structured objects in OWL: Representation and reasoning. Technical report, University of Oxford, UK, 2007.
14. Rolf Schwitter, Kaarel Kaljurand, Anne Cregan, Catherine Dolbear, and Glen Hart. A comparison of three controlled natural languages for owl 1.1. In Kendall G. Clark and Peter F. Patel-Schneider, editors, *Proceedings of OWL: Experiences and Directions (OWLED 2008 DC)*, Washington, DC (metro), April 2008.
15. A. Valente. *Legal Knowledge Engineering: A Modelling Approach*. PhD thesis, University of Amsterdam, Amsterdam, 1995.
16. A. Valente, J.A. Breuker, and P.W. Brouwer. Legal modelling and automated reasoning with ON-LINE. *International Journal of Human Computer Studies*, 51:1079–1126, 1999.
17. Andre Valente and Joost Breuker. ON-LINE: An architecture for modelling legal information. In *International Conference on Artificial Intelligence and Law (ICAIL-1995)*, pages 307–315, 1995.
18. Saskia van de Ven, Rinke Hoekstra, Radboud Winkels, Emile de Maat, and Ádám Kollar. MetaVex: Regulation drafting meets the semantic web. In Giovanni Sartor and Pompeu Casanovas, editors, *Computable Models of the Law. Languages, Dialogues, Games, Ontologies.*, volume 4884 of *Lecture Notes in Artificial Intelligence*, pages 42–55. Springer Verlag, 2008.
19. R. Winkels, A. Boer, and R. Hoekstra. ^{META}lex: An XML Standard for Legal Documents. In *Proceedings of the XML Europe Conference*, London (UK), 2003.