Making sense of legal texts

de Maat, E.

1 Introduction

A society is governed by its rules. Nowadays, many of those rules are codified in statutes, regulations and case law. In a sense, such rules provide a specification of how the society should be, how civilians in a society should behave, and how the society's government and civil servants should behave. Though regulations differ from technical specifications, and are written in natural language, rather than specified in a formal language or schematics, they are nonetheless not that different. Just like technical specifications conform to certain guidelines, there are extensive manuals governing legislative drafting, based upon traditions and experiences. Just like engineers, legislative drafters base their work on established best practices and design patterns. Because of this, the legislation is actually far more regular than one might expect from natural language texts.

Within the field of Computer Science & Law, the branch of computer science that deals with law, people attempt to make legislation more easily accessible, and uses this regularity of the legislation as a basis to deal with it. One way they do this is by creating portals that allow users to search for legislation, using keywords, titles, etc. This kind of application is widespread, and many legislative bodies provide such a portal, such as legislation.gov.uk in the United Kingdom, wetten.nl in the Netherlands, Norme-In-Rete[1] (Laws on Line) in Italy, LexML Brazil[2], the online version of the United States Code[3] and EUR-Lex[4], the European Union's database of regulations. In such portals, regulations are not stored as plain text, but in a more structured format, such as HTML, SGML or XML. This makes it possible to point users to a specific section of a document. Furthermore, there is often a lot of meta-data: information about the documents, such as date of enactment, date of repeal, jurisdiction, etc., each of which can help users to find the information they need.

However, for most users without a legal background, such applications are not helpful enough. Just like someone with a medical problem is looking for a diagnosis, and not just for medical information, a citizen is often looking for an answer to some legal problem, such as: Am I allowed to build a dormer on my house? For him, getting a link to the relevant regulations is an insufficient answer; he is looking for a simple yes or no. Thus, people within Computer Science & Law are also working on systems that provide these answers, by questioning the user about his situation and then applying the rules. These systems provide an answer to the question, as well as an explanation linked to the regulations.

A great variety of such systems have been created, covering a various topics and jurisdictions. A (random) selection includes a logic program covering the British Nationality Act (Sergot et al, 1986), the TAXMAN system, dealing with the United States' corporate tax law (McCarty & Sridharan, 1980, 1981), KRIP (Nitta & Nagao, 1986) and its successor KRIP-2 (Nitta, Nagao & Mizutori, 1988) focusing on Japanese patent law, ESPLEX (Biagioli, Mariani & Tiscornia, 1987), a system on agricultural tenancies in Italy, CLIME (Winkels et al., 1998) dealing with

[1] http://www.normeinrete.it/
[2] http://projeto.lexml.gov.br/
[3] http://uscode.house.gov/
[4] http://eur-lex.europa.eu/

ship classification (for among others access to ports and insurance), the XML encoding of the Ontario Freedom of Information and Protection of Privacy Act (Powers, Adler & Wishart, 2004) and the system for the Dutch Legal Aid Service Counter (van Engers et al, 2006).

Though not all of the systems above have been used by the users for whom they are intended, interest for such systems grows. This is evident from the fact that several systems have been created on the request of a government or company. In the document *Automated Assistance in Administrative Decision-Making*, the Australian government (2007) notes "these systems have the potential to improve the accuracy, consistency and transparency of administrative decision-making". Several companies make a living by developing and maintaining such systems, such as BeInformed (BeInformed Studio) in the Netherlands, Oracle (Ruleburst) in the United States and Knowledge Tools in Germany.

Each of these systems is based on a formal model of the law, though the precise formalism differs for each system. In the examples above, we already mentioned logic programming, XML, OWL and UML as possible modelling languages. We'll continue with some examples of such models.

The first one is taken from Sergot et al. (1986), who model the British Nationality Act. Their program attempts to determine whether or not someone is a British citizen. This example is about the first subsection of the act, namely:

British Nationality Act 1964, section 1.-(1)
A person born in the United Kingdom after commencement shall be a British citizen if at the time of birth his father or mother is
(a) a British citizen; or
(b) settled in the United Kingdom.

Sergot et al. have formalised the act in Prolog. For this subsection, without items (a) and (b), the formal representation reads[5]:

```
x is a British citizen
      if x was born in the U.K.
            and x was born on date y
            and y is after or on commencement
            and x has a parent who qualifies under 1.1 on date y
```

The first line forms the conclusion that can be drawn, namely that someone (represented by *x*) is a British citizen. The other lines form the conditions that must be fulfilled in order to draw that conclusion. The time of his birth (labelled *y*) must be after commencement, and at that time, his parents should qualify for either item (a) or item (b) from the original text. Sergot et al. have added an intermediate concept (`has a parent who qualifies under 1.1 on date y`) to represent this. Item (a) and (b) are modelled separately. The rules representing those items have the intermediate concept as their conclusion.

[5] This is not the final model used by Sergot et al., as it does not contain all the information they need for their complete program.

The representation for item (a) is:

```
x has a parent who qualifies under 1.1 on date y
      if z is a parent of x
            and z is a British citizen on date y
```

As said, the conclusion is the intermediate concept, while the condition is that a parent of *x* (labelled *z*) should be a British citizen on the appropriate date. Note that Sergot et al. have applied some common sense knowledge, and have modelled *father or mother* as *parent*.

To complete the model of subsection I.(-I), the model of item (b) is added, which is similar to that of item (a):

```
x has a parent who qualifies under 1.1 on date y
      if z is a parent of x
            and z is settled in the U.K. on date y
```

A representation that is somewhat further removed from natural language is given by Gordon (2008), who creates his models in the LKIF rule language. These rules can be used with the Carneades argument system to construct, evaluate and visualise arguments about a legal case. He gives an example from German family law:

§1601 BGB (Support Obligations)
Relatives in direct lineage are obligated to support each other.

Like the rules used by Sergot et al., the rules created by Gordon deal with a conclusion that follows from a set of conditions. Gordon's representation of the sentence given above is:

```
(rule §-1601-BGB
     (if (direct-lineage ?x ?y)
           (obligated-to-support ?x ?y)))
```

The first line is merely a name for the rule (which is a reference to the section of the law that is modelled). The second line gives the conditions, in terms of two people labelled *?x* and *?y*. The conditions states that if there is a `direct-lineage` relation between the two, the conclusion may be drawn, which is that there is also an `obligated-to-support` relation between them.

Another example, from van Engers et al., (2001), uses UML/OCL to model the Dutch income tax law. The example deals with the sentence:

Income Tax Act 2001, article 3.86, sub 2
The cycling deduction defined on the basis of the following subsections is reduced with the travelling expenses received for the travelling distance travelled by bicycle.

The model for this sentence is:

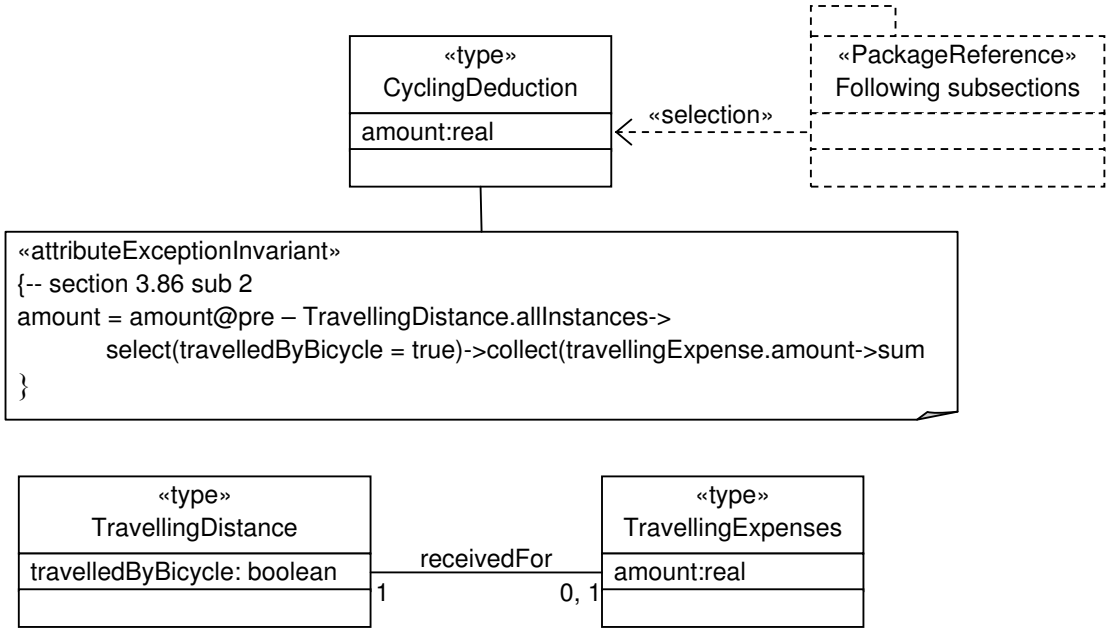

Figure 1: POWER UML/OCL model of cycling deduction

This model shows the different concepts from the sentence as a class: the *cycling deduction*, *travelling distance* and *travelling expenses*. The expenses and the deduction have a property *amount*, which indicates the amount of expenses and deduction. The distance has a property that indicates whether or not the distance was travelled by bicycle. There is a fourth concept, which is a reference to *the following subsections*, which need to be included separately, and which is represented in the model with dashed lines.

The distance and the expenses are related to each other, as the text deals with *travelling expenses received for the distance travelled*. This relation is indicated by a line between the two classes. Each distance can be associated with one amount of expenses, or not, which is indicated by a *0, 1* next to the line. Each amount of expenses must be associated with a distance travelled, which is indicated by a *1* next to the line.

The actual rule is given in OCL as a restriction related to the cycling deduction. It reads:

```
{-- section 3.86 sub 2
amount = amount@pre – TravellingDistance.allInstances->
      select(travelledByBicycle = true)->
      collect(travellingExpense.amount->sum
}
```

The first line is a reference to the text on which this rule is based. The text says that the cycling deduction is reduced with the travelling expenses that have been received, which is indicated by the first part of the rule: the (new) amount is the old amount (`amount@pre`) minus something. This something is collected as follows: take all travelling distances (i.e. all the trips the tax payer regularly makes), select from those distances the distances travelled by

bike, then select for those the amount of travelling expenses received, and finally sum these expenses.

Creating such complete, formal and executable representations of the law takes a lot of effort. This is a problem with any knowledge-based system, and has become known as the *knowledge acquisition bottleneck* (Feigenbaum, 1977). There is a lot of knowledge that has to be modelled, and often much of that knowledge is in the heads of experts. Moreover, the knowledge is often implicit, meaning that the experts themselves are not even aware of all that they know and use. Even when the information is documented or explained by the experts, much of it is left implicit, with the assumption that the receiver of the information is already aware of those parts. In addition, these experts are often valuable, and have limited availability. Likewise, the experts creating the model are valuable, making the creating of such model an expensive endeavour. We investigate ways to partially overcome this bottleneck for modelling the law.

In this thesis, we will focus on legislation. Legislation has the advantage that is has already been made explicit: it is already in text, rather than in the heads of experts. However, it is still a huge amount of knowledge that needs to be modelled. When it comes to legislation, two approaches are followed in order to reduce the effect of this bottleneck using computer applications: creating a model at the same time that the text is created and creating a model after the text has been created[6].

Creating a model of the text at the same time as the text requires specialised editing software. These editors require the user to provide information about the text they are writing at the same time as they are writing it. At the moment, editors that create a full, executable model of a text do not exist. There are, however, editors that help to create a document with sufficient metadata to allow for easy searching and linking. An example of such an editor is discussed in section 6.2.

Creating a model after the text is created can be done with automated methods of translating regulations, which concentrate on the recurring patterns that appear in the text. It has been attempted before in the POWER project (van Engers et al., 2000) and it successor (E)-POWER. Within this project, the proposal for a new Dutch Income Tax law was modelled in UML with OCL rules (see the example given in figure 1). This model was then used to simulate the effects of the new law, and to detect any inconsistencies. Within this project, a tool called OPAL (Object-oriented Parsing and Analysis of Legislation) was developed which creates models for noun phrases that appear in legislative texts using specific patterns (van Gog & van Engers, 2001), thus performing a small step of the modelling process automatically.

This method may seem to become obsolete, now that editors are being developed to create a model at the same time as the text is being drafted. Still, this is a valid approach to pursue. First of all, the editors are certainly not yet perfect, and even when they are, it will be some

[6] In theory, there is an obvious third approach, which is to create the model before the text is created, and then create the text based on the model instead of the other way around. Though there have been experiments with designing parts of legislation using decision tables (Overhoff and Molenaar, 1991), using such an approach for the entire law would require legislative drafters to acquire several new skills. Our focus is on methods that connect well to the current practice.

time before every legislative drafter has adopted them and each legislative document is drafted in this manner. Finally, these automated methods may actually be used within the editors, as import filters and as an additional component that recognises the text while the drafters are typing it, reducing the effort needed by the drafter to add all metadata to the document.

This second approach, automated methods of translating regulations, is the focus of the research presented here. The research is restricted to Dutch laws. Thus, the main question is:

To what extent can the creation of models of Dutch law be automated?

With Dutch law, we refer to Dutch law texts, not to the entire system of the law. In addition to relieving the acquisition bottleneck, a repeatable method for translation will also increase the inter-coder reliability of the modelling process. While the focus is on the (repeatable) automated translation, another aim of this research is to gain insight in the constructions used in legal texts.

For the translation process, we assume that the input is a Dutch legal text[7] without any metadata or layout (which often introduces implicit metadata). The output should, as much as is possible, be *isomorphic* with the original texts, as recommended by Bench-Capon and Coenen (1992). For a clear definition of isomorphism, they refer to Karpf (1989), who formulates the following conditions that a representation must fulfil in order to be isomorphic:

1. Each legal source is represented separately.
2. The representation preserves the structure of each legal source.
3. The representation preserves the traditional mutual relations, references and connections between the legal sources.
4. The representation of the legal sources and their mutual relations is separate from all other parts of the model, notably representation of queries and facts management.
5. If procedural law is part of the domain of the model then the law module will have representation of material as well as procedural rules and it is demanded that the whole system functions in accordance with and in the order following the procedural rules.

The translation from natural language to formal models is not done in a single step. Broadly speaking, it takes part in two phases. First, we need to make the structure of the document explicit, and then we can continue to model the meaning of the document.

Detecting the structure means that we identify the different parts of the document, such as chapters, articles and paragraphs, and then add metadata to the document to mark those parts. This serves a number of purposes:

- Identifying the different parts of the document allows for referring to those specific parts rather than to the document as a whole;
- Identifying the sentences is the first step in processing the actual text (as this is based on the grammatical structure of sentences);
- Separating headers and indexes from the actual content makes it possible to skip those headers and indexes when creating models of the content.

[7] As part of this research, the approach was only (systematically) tested using laws, though the same approach should also be valid for other legislative texts.

After the parts have been identified, we continue by searching for references in the text, which refer to the structure parts of the document itself and other documents. Like with the structure, metadata is added to the document to mark the references and to link them to their target. At this point, the document with metadata should be similar to the annotated documents used by portals such as wetten.nl (though without metadata on the history of the document). This means that the next steps can either be performed on documents that have been structured in this way, but also on (sufficiently) annotated documents from such portals.

For the creation of the actual model, we break the document apart into individual sentences, as identified when structuring the document. As we have seen in the examples above, many models are composed of partial models that represent individual sentences. This is also important in order to create an isomorphic model, as this means that the structure of the legislative source is preserved. As a preliminary step to the modelling, each sentence is classified. In the (E)-Power project this classification was left implicit and the step from the surface structure of sentences to a formal representation was typically too large to have useful automatic translations.

After a sentence has been classified, a model for that sentence is created, based on its classification, any signal words or language patterns that occur and (optionally) the grammatical parse of the sentence.

Finally, these individual models are combined into an integrated model. The different sentences in the law may refer to each other and to the same concepts. In this way, they are linked together. Thus, the models should be linked in the same way, by recognising when the sentences are linked and then creating a similar link between the individual models. This step involves a lot of common sense knowledge, which is not present in the text itself. Because of that, it cannot be (fully) automated.

Thus, by following these steps, we create a model of a legislative text. When such a model is coupled with (models of) knowledge from other sources, such as case law, legal tradition, juridical principles, etc., systems as mentioned above can be created. The entire process is depicted in figure 2.

In the next chapters, these steps will be further discussed.

Chapter 2 deals with the structure of Dutch laws, discussing the different elements that make up a law and the clues that the text contains to automatically detect these elements.

Chapter 3 discusses references in texts and the different textual forms they can take. Also discussed is a parser created to detect those references, and the way these references can be resolved, so that it is known to what they refer.

Chapter 4 proposes a classification of the provisions that occur in Dutch laws, and discusses two approaches on classifying provisions: a parser based on textual patterns, and an approach based upon machine-learning.

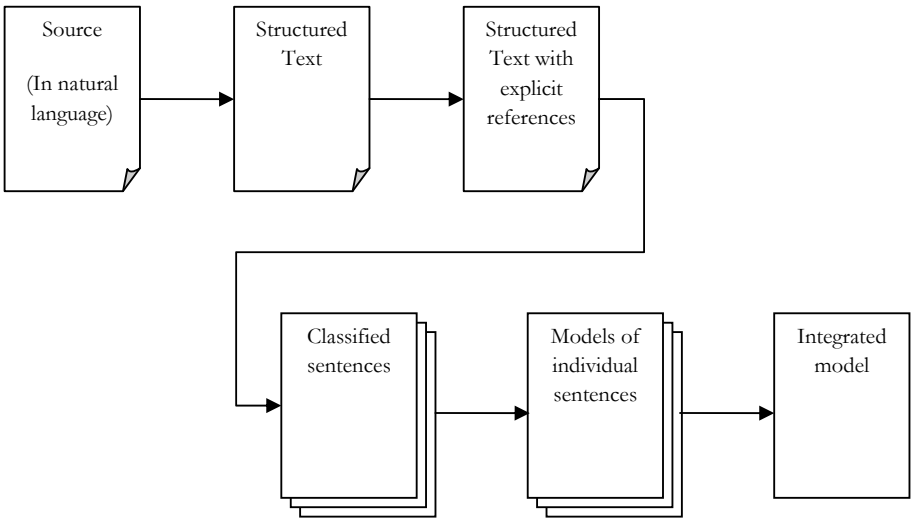

Figure 2: Steps in creating a model of a legislative text

Chapter 5 shows how a frame-based representation for provisions of each type can be automatically created, by splitting the text using either textual patterns or a full-fledged grammar parser for the Dutch language.

After this discussion of the entire process, some applications that are based on partial results of this process are described in chapter 6, followed by the conclusions.

The Dutch government publishes its regulations in Dutch only. Sometimes, a translation is created by the Ministry of Foreign affairs, but for most laws, no such translation is available. Unless noted otherwise, the translations provided in this text are created by the author to illustrate the research done, and are not official. Moreover, the translation that has been chosen is not always the most elegant. Instead, the aim has been to create translations whose structure resembles the structure of the original Dutch sentences.

The research presented in this book has been conducted over period of several years, and many partial results have been published before. In particular, parts of chapter 3 have been published before in de Maat, Winkels and van Engers (2006) and de Maat, Winkels and van Engers (2009). Parts of chapter 4 have been published in de Maat and Winkels (2007, 2008, 2009, 2010), de Maat, Winkels and van Engers (2009) and de Maat, Krabben and Winkels (2010). A part of chapter 5 has been published in de Maat and Winkels (2011a, 2011b). Parts of chapter 6 have appeared in Winkels et al. (2005) and de Maat, van de Ven, Winkels and van Engers (2009).