Making sense of legal texts

de Maat, E.

Publication date
2012

7 Conclusions and Future Work

In this thesis, we have presented methods to translate Dutch law texts to formal representations of that text.

Coming back to the question posed in the introduction:

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

We have broken the creation of such models into a number of different steps, namely:

1. structuring the law;
2. identifying references in the text;
3. classifying the different sentences;
4. generating models for the individual sentences;
5. integrating the separate models into a complete model.

When it comes to structuring a law and marking any references inside it, an automated approach can be very successful. By means of headings, the start (and the end) of elements in the body of the law is clearly marked. The introduction and the conclusion use fixed formulas which can be used to identify those parts of the text as well. For separating articles and subparagraphs into sentences, existing sentence splitting techniques seem to suffice. Only lists give some trouble, as different lists (and especially sub-lists) use different structures. As a result, lists require more patterns than the other elements, as well as more post-processing.

We have created a prototype parser based on regular expressions, which identifies the headers of chapters, articles, etc., the indexes of subparagraphs and lists and certain fixed formulas that appear in the text of a law. Due to time constraints, this prototype did not include full support for dealing with lists and quoted text. In an experiment (see section 2.8), this prototype parser achieved 100% accuracy on detecting the overall structure of ten different laws. For detecting the structure within each element, this prototype achieved an accuracy of 96%. All errors were related to lists.

Just as legislative texts are clearly structured, the references referring to those texts follow a clear structure as well. Elements in the text are generally referred to by the category of the element, such as chapter or article, and its index. Since the number of different categories used in the laws is limited, it is possible to find these combinations of category and index. More complicated, derived references have more than one target, combining a single category label with multiple indexes. Others use several steps to identify their target, using a category label and index combination for each step. However, these complicated structures are also limited and can be detected using regular expressions. Only the titles of laws cannot be detected using simple patterns, as they are too varied and too similar to regular text. A list of names is required to find those in the text. A parser based on these ideas was tested on six laws detected 97% of all the references in those texts correctly, with very few false positives (less than 0.1%).

All in all, it seems that the first steps of structuring a law and finding the references in it can be effectively done by a computer. The next step, the actual modelling, is significantly harder.

Recognising the type of a sentence is still a doable task. We have identified a number of different sentence types that occur in the law:

- obligations
- rights
- application provisions
- penalisations
- calculations
- delegation
- publication provisions
- definitions
- deeming provisions
- enactment date
- short title
- change provision (scope, insertion, replacement, repeal, renumbering)

Within each category, there are language patterns (mostly verb phrases) that are commonly used. There is only one exception. The official guidelines advise that obligations should preferably be written as a description of the desired situation, without using words like *must* or *should*. As a result, many of these sentences do not have a common language pattern.

We have created two classifiers to classify the sentences. One of them is pattern-based, and classifies sentences based upon the patterns found. If no pattern can be found, it defaults to classification. A second classifier is based on machine learning, using Support Vector Machines. Both classifiers achieve an accuracy of 90-95%.

For the actual modelling, we divide the sentences into two broad groups: sentences that deal with the law, and sentences that deal with the application domain of the law. Some sentences straddle these groups, for example consisting of a main sentence that deals with the law combined with an auxiliary sentence, such as a condition, that deals with the application domain.

The sentences that deal with the law cover a limited set of operations, and follow the same structure most of the time. Because of this structure, it is relatively easy to create a model of the law by filling the slots of a frame. A manual count has predicted that this will work for 96% of such sentences, provided that all references have been correctly identified, and disregarding any auxiliary sentences that deal with the application domain of the law, and that have to be handled separately.

The real hard task is the creation of models for those sentences that deal with the domain of the law. Using a parser, it is possible to identify the different elements of the sentence, and by doing so, create a model of the situation or activity described in the sentence. However, such norms need to be integrated, and this integration requires a lot of implicit knowledge that the computer does not (yet) have. Thus, we can conclude that automation is a realistic option for those sentences that deal with the law itself, but that for those sentences that deal with the actual domain of the law, an automated process remains limited to suggesting model

fragments for the individual sentences. Even with that restriction, though, an automated process can make useful contributions to the creation of portals for legislation and expert systems.

By recognising the structure of documents, and by adding a clear identification to each structure element, we make it possible to refer to the elements of a document. This makes navigating such documents easier, as an application can now redirect a user to the appropriate element, rather than leaving the user to search through the document by himself. Moreover, identifying the elements makes it possible to add metadata to the individual elements, which is useful for other applications. For example, in a law, different structure elements, such as articles may be enacted, changed and repeals at different dates. In such case, this information needs to be tracked for each structure element separate rather than for the law as a whole.

References refer to the documents and structure elements of the documents. By automatically recognising and resolving the references, we make the "web" of the law explicit, making it easier for the user to navigate that web. When a reference has been marked-up, a computer application can automatically present the correct document, which means the user does not have to search for the document by hand (and, as mentioned above, if the target's document structure has been identified, then the application can even send the user to the appropriate structure element as well). In addition, it is possible to follow the references the other way: an application can show what other regulations refer to the document that is currently viewed by the user.

A reference parser is not only valuable in detecting references in documents in which references have not yet been annotated. In existing collections, documents may have references annotated with links to other documents within that collection. Applying a generic reference parser can help to link such documents to legislation outside that collection, effectively linking multiple collections together (see Winkels et al., 2005).

Making the web of the law explicit also opens the door for other research. For example, the number of references in a law can be considered as an indication of the complexity of that law (Bourcier & Mazzega, 2007). When the references are made explicit, this measure of complexity can more easily be determined. Also, using notions from graph theory, it is possible to draw other conclusions with regard to this web of the law, such as identifying key articles (see Smith, 2005, Mazzega, Bourcier & Boulet, 2009, Bommarito, Katz & Zelner, 2009 and Liiv, Vedeshin, & Täks, 2007).

The categorisation of sentences is intended as a preparatory step for the creation of models. However, the categorisation itself can further support a user that is searching through the documents. It allows a user to search through a document in a simpler manner. For example, if a user is searching for specific definition, an application could show only the definitions. If a user is reading the law in order to find about the norms contained in it, the application could hide all modifying provisions[180].

[180] This corresponds with the manner in which wetten.nl presents Dutch regulations; on this portal, all modifying provisions have been removed from the regulations and are not displayed.

The search functions are further improved by the frames that are generated, as those frames make it possible to search for a definition of a specific term or for modifications to a specific location. Other frames, such as setting the citation title, can be used to generate appropriate metadata for the document. The frames that are generated for modifying sentences contain sufficient information to generate consolidated versions of the legislation that has been changed by them (see Palmirani & Brighi, 2002, Ogawa, Inagaki & Toyama, 2008, de Maat et al., 2009).

Finally, as was the original intention, the frames can be used as a basis for the model of an expert system. The automated generation of models for individual sentences, and possibly, the automated partial integration of such sentences can reduce the effort needed to create complete models, and will improve the uniformity of such models.

So, in various ways, the automated processing of legislation can improve access to the law and increase the quality of the law.

7.1 Regularity of Legislative Sources

The success of this method hinges on the regularity of the legislative texts. As a result of tradition and guidance through the official guidelines, the Dutch legislative texts are very structured. Still, there are a number of issues where a stricter adherence to the guidelines, or small additions to the guidelines, could improve the (automated) understanding of the text.

For the first step, the recognition of the structure of documents, the parsers can be simplified if new legislation coheres more closely to the official guidelines. Specifically, adhering to the following guidelines would help:

- Use the correct order of the different levels in a law (section, division, title, chapter, part and book).
- End all list items in a semi-colon, except for the last one, which ends in a full-stop if the list has no conclusion.
- Use the correct format for the indexes.

On other issues, the guidelines could be expanded in order to facilitate the comprehension of the texts. First of all, it would help to add punctuation to denote the end of quoted structure elements (i.e. entire list elements, paragraphs, articles, chapters, etc.) Likewise, marking the end of sub-lists using some dedicated punctuation also eliminates problems with the detection of the end of a sub-list. Sub-headers are also difficult to detect, and not using such sub-headers would reduce the chance of errors in the recognition of the document structure. Finally, the use of a colon at the end of an introduction of a list makes it easier to recognise the list, and avoids the chance of errors with lists. With regard to references, the current practices do not interfere with automated recognition. There are some occurrences of references that deviate from the common practice, but nothing that warrants new guidelines. However, when it comes to resolving the references, there is one type of reference that is difficult to resolve correctly: the range. The problem with a reference to a range is that it cannot simply be resolved by knowing the reference; you also need to know the structure of the document being referred to. For example, a reference to *article 41* is simply a reference to *article 41*. A reference to *articles 41 to 44* may be a reference to *articles 41, 42, 43 and 44* or *articles 41, 42, 43,*

43a and 44, etc. This also means that a range cannot be permanently resolved, as its set of targets may change over time.

The fact that the set of targets may change over time does not only make it harder to create an easily maintained computer model of the text, but also increases the chance of introducing errors. A newly created article may be included in a reference unintended. The same is true for references that refer to *the previous article, the previous subparagraph*, etc. After the text has been changed, these references may no longer refer to their intended targets. Thus, to avoid errors, it may be preferable to avoid references to ranges and relative references.

When it comes to categorisation of the sentences, the Dutch legislation has one significant advantage to some other jurisdictions, and that is that the rules in a law are separated from the motivation for such rules. As a result, no steps need to be taken to separate rules and motivations. On the other hand, in many jurisdictions, clear signal words like *ought*, *must* or *should* are used in obligations, whereas the Dutch guidelines advise against using such words. Because of this, no clear signal words are present in obligations, making them harder to detect.

Though such changes in the legislative texts themselves would increase the performance of the processing by a computer (and, perhaps, processing by humans as well, though additional research will be needed to determine if this is actually the case), the largest improvement is still to be gained by further implementing the parsers and other tools presented in this book.

First of all, not all of the ideas presented in this book have been implemented. The structure parser needs to be extended so that it can better deal with lists, including the separation of list items and sub-list items and the recognition of the conclusion of a list. It also needs to be extended in order to recognise quoted text elements (which, in Dutch legislation, is not marked using quotation marks) and to recognise appendices.

The sentence classifier would perform significantly better on lists if each list item was first converted into a sentence (by combining it with the list introduction and the list conclusion, if it exists) and then classified. We tested approaches that limited themselves to either the introduction or the list item, but this lead to errors that should be prevented by the conversion proposed. Furthermore, since the sentence classifier is supposed to classify the main sentence, it would be improved by basing it on main sentences only, leaving auxiliary sentences out. The current classifiers, which do not leave the auxiliary sentences out will sometimes misclassify the sentence based on patterns or keywords appearing in an auxiliary sentence.

7.2 Architecture for an Integrated Application

Next to completing the current tools, the entire process can be improved by creating a more integrated application. Currently, the different tasks that are needed to go from a natural language text to different models are all achieved by different applications, which have to be run by hand in order to achieve a full conversion. The prototypes that have been developed during the course of this research are:

- a structure parser, consisting of various GATE modules and a post-processor in JAVA;
- a reference parser, built using the JAVA Compiler Compiler (JAVACC);

- a sentence classifier, built in JAVA, using the JAVA regular expressions library;
- a model generator for norms and definitions, also built in JAVA, using XML models generated by the Alpino parser as input.

At the moment, no model generator for sentences other than definitions and norms exists. The Alpino parser is based on a Prolog grammar, and can be run on a Linux machine.

Obviously, running all these programs by hand is not a good setup for an automated system. It would be desirable to have a pipeline of integrated modules rather than a set of separate applications. A good way to achieve such a pipeline would be by converting all modules that have been developed to GATE modules. GATE is specifically developed for natural language processing tasks, and is supported by an active community.

As mentioned, the structure parser has already been implemented using various modules from GATE combined with new JAPE rules. The reference parser can also easily be converted to JAPE rules, as it is mostly based on different patterns. This has a small disadvantage, since JAPE is based on regular expressions instead of a grammar. List items can be nested, and since regular expressions are not recursive, it is not possible to support any level of nesting using JAPE. This is not a big problem, however, since a fixed maximum depth of three or four should be sufficient to cover most, if not all, references.

Likewise, the patterns used for the sentence classifier can be converted to JAPE rules, making it possible to include the sentence classifier as a JAPE transducer inside the GATE framework.

A parser like Alpino cannot be converted to JAPE regular expressions. However, GATE makes it possible to include functions as plug-ins using wrapper classes. In this way, GATE includes several parsers for the English language. Assuming a parser for Dutch becomes available for GATE, we can then include the model generator as a set of JAPE rules on top of the annotations made by the parser. Models for sentences that do not require the parser but can be generated using language patterns can also be generated using JAPE, with rules again based on the actual words instead of the parser's annotations. The frames as presented in chapter 5 will be included as annotations in the document.

So, if all these conversion are made, a GATE pipeline can be constructed using the following processing modules in order, mimicking the process as it has been with the individual applications:

1. tokeniser;
2. JAPE Transducer which corrects tokens which are part of an index that mixes numbers, letters and punctuation;
3. sentence splitter;
4. JAPE Transducer to identify structural elements;
5. JAPE Transducer to identify references;
6. JAPE Transducer to classify the sentences;
7. Dutch parsing module(s);
8. JAPE Transducer to generate the frame annotations.

After the GATE modules have been run, a postprocessor can take care of those tasks that remain: add containers to the output (see chapter 2) and resolve the references to proper URIs (see chapter 3). The result of this process will be a XML file in which the structure is marked, references are marked, and elements of each sentence are marked according to their role in the sentence (in terms of the legal meaning, not their semantic role or syntactical role)(see chapter 4). This XML document can later be used to generate models in a specific formal language.

Running the postprocessors last, rather than in-between steps, also means that they can profit from the additional information found. The postprocessor for adding containers benefits from the knowledge of which sentences are modifying sentences, as those may introduce quoted text elements. The postprocessor for resolving references needs to know which sentences are scope provisions, as this will affect the references made following that scope provision.

As said, this pipeline follows the process as it has been executed using the individual applications. It can be streamlined a bit further by creating a dedicated tokeniser that already recognises the indexes, so that no corrections are needed. Furthermore, since it is desirable to recognise subordinate sentences before classifying sentences in order to avoid confusing due to patterns appearing inside these subordinate sentences (see chapter 4), the parsing modules can be moved to before the classifying module, as parsing the sentence will also identify subordinate sentences, resulting in the following GATE pipeline:

1. "legal" tokeniser;
2. sentence splitter;
3. JAPE Transducer to identify structural elements;
4. JAPE Transducer to identify references;
5. Dutch parsing module(s);
6. JAPE Transducer to classify the sentences;
7. JAPE Transducer to generate the frame annotations.

Again, the GATE pipeline is followed by a postprocessor.

There are two issues with the Dutch parsing module that are relevant for this automatic processing pipeline. First of all, the Alpino parser tends to have problems with references that contain a lot of punctuation. It is expected that this holds true for any parser. To avoid this, any references (which have already been recognised) should be annotated in the input of the parser so that it does not attempt to analyse it (but simply regards the entire reference as a name).

The second issue is that a parser usually generates multiple parses for each input sentence. When including it in a pipeline, only one parse will be considered (i.e. the one that is considered "best"). This may not be the correct parse. In order for the pipeline (and the automated processing) to function properly, the preferred parse should usually be the correct parse.

As it is unlikely that the preferred parse will always be the correct parse, a second tool should be available next to this main pipeline, which allows users to correct any errors introduced because the preferred parse was not the correct parse. The tool uses the XML file that is the output of the main pipeline as input. Users can select sentences that are not modelled correctly. These sentences are then put through a smaller version of the pipeline, consisting of

variants the Dutch parsing module(s) and the transducer to generate frame annotations. The variant of the parsing modules should return multiple parses[181] (i.e. the top ten, or some other fixed number, rather than only the preferred parse). The transducer is run once for each parse, thus generating multiple models. The user should be able to select the best model (or even annotate the sentence manually), which is then saved in the XML file.

There is an overlap in the patterns searched for by both the reference parser and the structure parser, as the label and the number as they appear in a header are the same as they appear in a reference parser. With regards to performance, it may be beneficial to combine this search. This can be achieved by switching the order of the reference parser and the structure parser. The reference parser will find both the references as the label/number combinations that appear in the header. The structure parser can then use the references that have been identified as input to find the headers, changing the annotation from "reference" to "header".

7.3 Future Expansions

As discussed in the previous paragraphs, this method still requires a lot of fine-tuning. More importantly, it also requires more testing. This is especially true of the actual modelling steps, which have not yet been tested systematically. Without doubt, this will result in more issues to be addressed.

In addition to the fine-tuning, though, the current method can be extended beyond the ideas presented here. There are two important directions for such expansions:

1. The current method is aimed at laws, and should be expanded to include other regulations and case law.
2. The current method has not explored all possibilities for modelling sentences that deal with the domain of the law, and it may be possible to generate more specific models.

As for the extension of the range of legislative sources covered, this is something that mostly comes down to expanding the different tools with new patterns for the additional documents. All (national) legislation in the Netherlands falls under the guidelines, and thus should be using similar language and structure. The structure parser has already been tested on a ministerial decree, and with some additions to detect the introduction of such a document, which differs from that for a law, the parser was able to handle this decree.

Extending the reference parser to handle references to other legislation mostly comes down to adding the names of such legislation to the list of names of laws. As lower regulations often have a similar structure to a law, the references to them also follow the same structure as references to the law. Still, it is likely that some new patterns emerge and have to be added.

Here, it may be interesting to include other documents that refer to legislation, such as case law and doctrine. These documents will refer to legislation in much the same way as legislation refers to legislation, though there are a few differences. First of all, these documents are much more likely to use informal names for legislation, which, like the formal names, will have to be collected in a list before they can be properly detected. Secondly, any incomplete references in

[181] Note that GATE annotations differ from XML annotations in that they can overlap; this makes it possible to have multiple parses annotated within one document.

such a document will have a different meaning. A reference to *article 9* in a regulation refers to "article 9 of this regulation", while in case law or doctrine, it refers to "article 9 of the regulation that is the discussion of this (section of the) text". Thus, the manner in which such references are resolved needs to be changed, and will be more complicated.

Finally, the sentence classifier and the method for generating model fragments should perform the same on the sentences that appear inside legislation that appear in regulations other than laws with regard to the types of sentences already included. However, it may be that other types of sentences exist in those documents, and the methods need to be expanded to cover those.

As for expanding the modelling, the current method treats all prepositional phrases (that are not linked to the direct object or indirect object) in the same way: as a general modifier of the action described in the sentence. In order to make the models more informative, it would be interesting to somehow classify these phrases in categories, such as temporal or spatial constraints. Research into this area is already being conducted (see for example O'Hara & Wiebe, 2003). Such a classification will allow for a more meaningful model of these phrases.

Looking at the core of the sentences, there is also room for some more specific modelling. In the current method, only two types of norms are distinguished: rights and obligations. Most theories on norms distinguish several other categories. Possibly the largest generalisation made in our simple division is that it treats power-conferring norms the same as behaviour regulating norms. In doing so, it ignores the fact that a power-conferring norm does not only confer a certain ability, but a certain responsibility as well.

In order to treat power-conferring norms different from other norms, we need to be able to distinguish them. Our current methods for classification, using language patterns or machine learning based on the words appearing in the text cannot make this classification. However, it may be possible to recognise the power-conferring norms after the sentence has been parsed, and additional information can be added to the analysis, such as the agent of the action. Since a power-conferring norm will (generally) target a civil servant or government institute and not a civilian, such information may help make a more advanced distinction between the different types of norms. However, even when classification is made more precise, power-conferring norms will likely remain a troublesome category, as they often are broad statements, which are difficult to integrate with more precise statements.

There seem to be few (practical) expert systems that focus on these norms. This seems understandable. The broad terms used in these norms do not lend themselves well to a computer application. Furthermore, there will be few routine questions that involve these norms (and an expert system is most useful for routine questions). Hence, it remains to be seen whether it is useful to focus on these power-conferring norms.

Next to the power conferring norms, the procedural norms also deserve more attention. For the norms that form a procedure together, the order in which they appear in the text is important, as the steps in the procedure have to be followed in that order. Currently, the method does not include any patterns or other handholds to discover whether a norm is part of a procedure, nor any ways to model the outcome. The same is true for procedural

calculations, though those are easier to recognise, as any calculations that try to establish the same value and apply at the same time are usually part of a procedure and should be applied in order.

These two extensions are aimed at improving the method presented in this thesis. However, there also needs to be more attention to the use of the metadata added by the different tools discussed here. Currently, such data is not offered to the users in a manner that allows them to utilise it to the fullest. For example, Hoekstra (2011) argues that the current Dutch government portal for legislation, wetten.nl, presents its information in a too restricted way:

> Wetten.nl presents regulations as books with hyperlinks; the position of an article within the running text of a regulation is the only context provided. Given the highly networked structure of legislation, this traditional restricted presentation is suboptimal: potential alternative ways of serialising one or more regulation texts (e.g. by topic) are discarded. This is not only a potential problem for businesses and citizens trying to understand the norms applying to their case, it is problematic for the civil servants and government organisations that have to apply these norms as well.

Hoekstra shows how the current contents of the portal can be published as "5-star open data", which should be more flexible. Such an approach will need to be elaborated upon in order to accommodate the additional data (such as sentence types) we can extract.

7.4 Text First or Model First?

In the introduction, it was mentioned that this method is one of three methods that are being explored with regards to the creation of models of legislation. The other two methods are:

1. Creating the model at the same time that the legislative text is drafted, using a specialised editor;
2. Creating the model first, and then generating the legislative text from the model.

Just like parsing, the editor allows legal drafters to create a law using natural language, whereas starting with the model does not. So, the main question becomes: is it better to specify the law in a natural language or in a formal language?

An important difference between these two approaches is, of course, the output. The processes each have two outputs, two representations of the law, one in natural language and one in formal language. When starting with natural language, we will end up with a good and correct text, but it costs some effort to create a correct computer model. Starting out with the model will give us a correct computer model, but it will create some effort to create a human readable text.

In itself, generating a natural language description of a formal model is doable; each statement in a formal language can be described in a sentence in natural language. However, a collection of such statements does not yet form a complete and readable text. Ruiter (1987) mentions that a law could be written in "legal normative sentences", with each sentence including a normative operator (such as "may" or "must"), the norm addressee, the behaviour and conditions. Such legal normative sentences are quite close to a description of a formal model – but Ruiter states that if a law were completely composed of such sentences, they would be tedious, complicated and full of unnecessary repetitions. So, in order to generate an actual text

we would need to integrate those sentences, which would involve removing unnecessary repetitions. Also, the common sense knowledge that needs to be added to a legal text when creating a formal model needs to be removed when creating a legal text from such a formal model.

In both approaches, additional effort will be required to generate the "translation". We have already seen that the integration of generated model fragments requires human intervention; it is likely that the same is true for the integration of generated sentences.

The main difference between the two approaches will be that the quality of the original specification, either in natural language or formal language, will be higher than that of its translation. So, the question is: which one do we want to see as "leading"? We should ensure that the leading specification is the one that has the highest quality.

Opting for natural language seems to be the better choice. Even though formal languages retain snippets of natural language, they are not as rich as natural language, which means that there will always be things we cannot express in a formal language that we can express in natural language. So, in general, starting with natural language seems is preferable.

Still, there are situations in which starting with more formal models may be preferable, such as writing amending provisions, where an approach that starts out with making the actual modifications wins out over describing them in text. Likewise, Voermans (1998) concludes that in some cases, a more formal model, like a decision tree, is better. In such cases, using a formal model is more desirable than using natural language – in fact, it would be better not to translate the formal model to natural language, as the formal model is better at communicating the information than the natural language translation.

So, in the end, a mix of both approaches is best. Situations that are difficult to explain clearly in natural language should be drafted and presented in a more formal language. Other elements of the law should be drafted in natural language and then translated to a formal model – preferably, as much as possible, by the computer itself.