Making sense of legal texts

de Maat, E.

6 Applications

In this chapter, we will discuss two applications that have been created that use the research that has been presented in the previous chapters. The Semantic Network for the Dutch Tax and Customs Administration is a content management system intended to deal with large amounts of sources of law which have been provided by various different publishers. It uses parsers to extract useful metadata. The second application is MetaVex, an editor for legal sources. It uses language patterns for templates and to generate natural language texts from models.

6.1 Semantic Network for the Dutch Tax and Customs Administration

The Dutch Tax and Customs Administration (DTCA), like many other organisations, deals with a multitude of (digital) legal data. Part of this data is published by themselves, but other data is purchased from publishers. They run into a number of different problems with these collections. First of all, each publisher uses its own format for its collection, and the DTCA adds to that collection using that format. Should the DTCA wish to switch to a different publisher, they will need to translate their corporate data to the format of that new publisher. Secondly, documents from the same collection have marked-up interrelations, but they are not related to documents from other collections. These relations are added by the DTCA. This is a time and effort consuming task, which needs to be repeated every time one of the collections is updated. Finally, there is a lot of redundancy in the total collection. Most publishers will include the relevant legislation in their collection, which means it turns up several times in the total collection. This can be confusing for the employees of the DTCA, but also means that the DTCA may be able to save on costs if they can avoid buying this duplicate data.

To counter these problems, we have built a prototype content management system (CMS) (see Winkels, Boer, de Maat, van Engers, Breebaart, & Melger, 2005) based on MetaLex XML. In this system, metadata about the documents in the DTCA collection is stored outside these databases using RDF. This metadata is stored at the expression level instead of the item level (see chapter 3).

For each expression in the collection, the CMS contains a RDF representation of the expression and its structure. For example, if the CMS contains the Income Tax Law as it was modified on January 1st, 2011, then the RDF graph contains a node for that version of the Income Tax Law, nodes for each chapter of that version of the Income Tax Law (which are linked to the node representing the entire law), nodes for each article in that version of the law (which are linked to the node representing the chapter they are a part of), etc.

Metadata is linked to these RDF nodes. In particular, references between two documents are stored as a reference between two expressions. The documents in the collection are each linked to the expression they exemplify. Thus, any document is automatically linked to all the metadata that is available for the expression (instead of just the metadata that is part of the collection it is in). If a new document is added to the total collection, it merely needs to be linked to the appropriate node in the RDF graph, which will link it to all relevant metadata.

In order to create and maintain this network a number of parsers were created, which extracted the relevant metadata.

First of all, the identity of each document had to be established. To do so, we need to know the type of document (legislation, part of legislation, case law or legal commentary), the title or name of the document and the version of the document. There are two sources for this information: the actual text of the document and any metadata added by the publisher. Obviously, simply extracting the metadata is the easiest and most reliable method. This information is structured and has already been checked by human experts at the publisher. All the documents used for the prototype included such metadata.

If the metadata is lacking, we can establish the identity using other information. If the publisher has used meaningful filenames or addresses, we may be able to identify the document by means of this filename. If not, we can study the document text itself to see if it contains a title. As some documents in a collection represent only a part of a law (e.g. a single article), this is not always the case. Finally, we may be able to derive the information from the text other documents use to refer to this document (using links provided by the publisher.)

The next step is to find the reference in the texts. Within each collection, documents will often have links to other documents in the collection, but will lack mark-up for references to documents that are not part of the collection. A predecessor of the parser described in chapter 3 was used to detect those references.

In this manner, we can provide the DTCA with a knowledge base of the field they work in that is both format-independent and publisher independent. Other applications used by the DTCA, either for themselves or their clients, can connect to this database with a static link to the correct concept in the network.

Other research, such as the work of Urbani, Kotoulas, Maassen, van Harmelen and Bal (2010), who have worked with a dataset of 100 billion triples (which are the links between RDF nodes) shows that this solution will scale up to the many sources it will have to deal with in the future.

6.2 MetaVex

An important part of the work done by ministries, national parliaments, city councils, etc. consists of the drafting of new legislation and of amendments to existing legislation. Much of this work is done by means of general purpose word-processing software. Such generic software, however, does not include support for the legislative process, and is not well-integrated with the systems that are used to store, publish and search through legal documents.

At the Leibniz Center for Law, an editor for legislative drafters has been created with which they can create laws in CEN/MetaLex XML format, which provides better integration with other systems than generic document formats do (see van de Ven, Hoekstra, Winkels, de Maat & Kollar, 2008). This editor has a look and feel similar to normal word processors, and

document editing is done in a WYSIWYG[173] interface. Legislative drafters do not need any knowledge of the underlying XML in order to create new documents. The editor ensures that the output is always a valid XML document.

By means of templates, the editor attempts to speed up the process of generating document structure and content, and supports the user in complying with the official guidelines. These templates are based on the official guidelines and on other patterns described in chapter 4.

MetaVex allows drafters to mark-up references to other sources, but also shows them all the references that point to the document they are editing, so that they can more easily determine the impact of their changes.

More elaborate support is available for the creation of amending documents. This is an important topic, as the majority of laws that are produced are in fact amending laws (i.e. laws that change an existing law rather than introducing completely new legislation). In addition, each law that passes through Parliament will give rise to a number of amendments proposed by members of Parliament.

In many ways, amending documents are more challenging to handle than regular documents. They contain descriptions of the changes to be made rather than the complete new text. Because of this, the actual impact of the new text can sometimes be difficult to perceive. To help the reader with such documents, there are applications that show the new text next to the original text, with any modifications marked using highlights and strikethroughs. The European Parliament makes use of this method, and the Italian Senate has also adopted a tool, called TafWeb, that shows this comparison (see Bacci, Spinosa, Marchetti & Battistoni, 2009).

MetaVex is intended to help the drafters of an amending document in a similar way. It does so by allowing a legislative drafter to make changes to a copy of the original act. The system then compares this modified copy with the original document, detects the changes and represents them in natural language (see de Maat, van de Ven, Winkels & van Engers, 2009).

This approach is similar to that of the *Themis* system used in Tasmania (Arnold-Moore, 1997). In that system, the original acts are stored in SGML. When a user wishes to create an amending law, he checks the act out. At this point, the act is converted to a MS Word document. The user then uses MS Word to edit this document. The Themis system includes dedicated MS Word macros, templates and toolbars to assist the user in editing the document. When the user has finished making modifications, he saves the changes in a Rich Text Format file. In this file, the changes are marked using underlines for insertions and strikethroughs for deletions. Based on this file, two new SGML files are generated: one representing the original act, and one representing the modified act. These two SGML files are then compared in order to compile a list of changes. Compared to the Themis system, MetaVex has the advantage that it is a native XML editor, so we do not need to convert the XML file to a different format and then back to XML again.

As mentioned above, the modified act is compared to the original document. This is done in two steps. First of all, the structure of the modified document is compared to the structure of

[173] What you see is what you get.

the original document. This is made possible through the fact that the structure is marked-up using MetaLex. This comparison detects three types of modifications:

1. Deletion of an entire element: if an element with a specific ID does exist in the original document, but not in the modified document, it has been deleted.
2. Insertion of an entire element: if an element with a specific ID does exist in the modified document, but not in the original document, it has been inserted as a new element[174].
3. Relocation of an entire element: if an element with a specific ID occurs in a different position in the modified document than in the original document, it has been moved. Though we can detect such a change, this is not an operation that is applied in Dutch legislation, as is evident from the fact that there is no sentence type describing a relocation.

After this structural comparison has been made, all the elements in the modified document that have not been newly inserted are compared with the original text to find any textual changes, using a *diff* function[175]. This process will also detect renumbering, as this is a modification of the text of the index (which is tagged separately in MetaLex).

Some actions by the user can confuse this system and cause it to come to the wrong conclusion. For example, a user could copy an article and delete the original. In effect, this is a move of the article, but because the copy has a different ID than the original article, MetaVex will detect a deletion and an insertion. However, for most common actions, this procedure adequately detects the changes.

After all changes have been detected, metadata describing these changes are stored in an XML file. Now, an amending text can be generated, which describes the changes in natural language. The user can later add additional texts, such as a motivation or provisions that do not modify existing legislation. For each insertion, deletion, renumbering, etc. a single sentence is generated describing that modification. These changes are described using the sentence formats that are described in section 4.4.3 as templates. Using the patterns described there, any change can be expressed in natural language by taking an appropriate pattern and fill it out in the right way. For example, the template for a textual replacement is:

> In [*target*], «[*old_text*]» is replaced by: [*new_text*].

The texts in square brackets indicate fields that have to be filled out. So, if a user has modified article 7 (target), replacing the words *director* (old text) with *board of directors* (new text), then this information has been detected using the steps described above. Filling out the fields in the template results in:

> In article 7, «director» is replaced by: board of directors.

---

[174] MetaVex automatically assigns a new ID to each new element.
[175] To limit the number of diff operations, MetaVex flags all elements that have been edited by the user in some way. When searching for changes, MetaVex only checks the elements that have been flagged in this way.

Effectively, this is the reverse of the process we used in section 5.1.4 to extract the information for a frame from a sentence.

The location of the modification is derived from the MetaLex document, in which the category and index of each element is marked. For example, if there is a change in a list item with index "u", which is part of a paragraph with index "1", which in turn is part of an article with index "21", then the entire natural language description would become *artikel 21, eerste lid, onderdeel u* (lit.: article 21, first paragraph, item u). Basically, this is simply the concatenation of the different category labels and indexes, though some modifications are needed. As mentioned in section 2.4, in Dutch laws, the paragraphs and list items do not have a full heading, but are only numbered. This means that the category labels have to be added. Furthermore, in a reference to a paragraph, the index number is replaced by an ordinal number (e.g. *first* instead of *1*), so this has to be converted.

The other necessary texts to fill out the "blanks" in the sentences are simply those texts that have been identified when detecting the changes (i.e. the texts that are deleted or inserted).

This approach works, but does sometimes yield rather ugly constructions, as there is a complete sentence generated for each change. Sometimes, it is more desirable to group these changes, which requires a more sophisticated approach. First of all, generating individual sentence may result in something like:

Article I
In Article 12, sub 1, … is replaced by … .

Article II
In Article 12, sub 2, … is replaced by ….

Article III
Article 12, sub 3 is repealed.

Though these sentences correctly describe the modifications, it is not how these changes are usually represented in an amending law. If there are several changes in the same part of the document, they are usually grouped, and preceded by a scope declaration as described in section 4.4.3. Any further references are then made relative to that scope. So, in the case of the example above, the preferred way of presenting the modifications would be:

Article I
Article 12 is modified as follows:
A. In sub 1, … is replaced by …
B. In sub 2, … is replaced by …
C. Sub 3 is repealed.

To achieve this result, the changes must be grouped according to location before the text is generated[176]. If a location contains more than one change, a grouping construct as shown above must be used.

Another correct, but ugly representation occurs when the same modification is repeated several times at different locations, such as:

Article I
A. In Article 1, «the director» is replaced by: the board of directors.
B. In Article 3, «the director» is replaced by: the board of directors.
C. In Article 4, «the director» is replaced by: the board of directors.

This will happen when a reference that is used multiple times needs to be updated, or when a concept needs to be replaced by some other concept. In these cases, the modifications need to be grouped, too, in a sentence like[177]:

Chapter 7 is modified as follows:
All occurrences of «the director» are replaced by: the board of directors.

In order to perform this grouping, we need to compare the texts of each (similar) modification, in order to determine whether a modification is repeated multiple times. If we wish to come close to the sentence a human expert would produce, we should also group the indexes[178]. For example, if articles 112, 113, 114, 115 and 116 are affected, it would usually be described as *articles 112-116* or *chapter 12*, if these chapters combined form chapter 12 (or if the other articles in chapter 12 would not be affected by this modification). When making such a grouping, we need to check if there are no other articles in that range (e.g. an article 114a) that should not be affected.

Next to these problems with grouping, there is another way in which this system produces sentences that describe the modifications correctly, but not in the desired way. When a change is made inside a sentence, we know which words have been removed and which have been added, and we can describe those modifications in natural language. However, the Dutch guidelines prescribe that only complete *meaningful units* may be replaced. Thus, if *Minister of Foreign Affairs* is replaced by *Minister of Internal Affairs*, it should be stated that way, and not by stating that *foreign* is replaced by *internal*. At this moment, MetaVex does not have enough knowledge of what constitutes a meaningful unit; incorporating a chunker or parser to split the sentence could help here.

A final situation in which the computer generated texts may differ from a text written by a human expert is in the description of locations in the text. As mentioned above, the computer generates descriptions of locations by concatenating the categories and indexes of each containing element. So, the description for article 34, contained in section 4 of chapter 1

[176] MetaVex does groups the changes immediately after detecting them, so this step is not actually made when generating the text, but already before.
[177] Textual modifications will only be grouped if they are exactly the same. If there is a difference in capitalisation, for example, they cannot all be grouped together.
[178] The question is whether we actually want to replace the enumerations with such ranges; as mentioned in section 2.8, using ranges may result in errors.

would be described as *Chapter 1, section 4, article 34*. However, it is common for articles to have unique number (i.e. the numbering does not reset at the start of each new chapter, or each index includes the number of the chapter). If this is the case, referring to *article 34* is sufficient. In order to generate this shortened description, we need to determine whether or not the articles are continuously numbered, either within the entire law, or within some lower level. We can determine this by first studying all article indexes; if they are different, it is fair to assume that they are continuously numbered. If not, we can repeat the procedure at lower levels to determine if they are continuously numbered at that level. After we have determined if the level at which the articles are continuously numbered, we can then proceed to generate the appropriate description, leaving out any levels that do not need to be mentioned in order to get a precise location.

MetaVex collects a formal description of all the modifications made to an act, so we can use that information to generate a consolidated version of that act, i.e. the original version of the act with all the changes applied to it. A similar system has been developed for Italian acts by Palmirani and Brighi (2002) who created a system that can semi-automatically create consolidations based upon amendment acts that have already been marked-up before. Ogawa, Inagaki and Toyama (2008) have created a system that can consolidate Japanese acts by first parsing the amending clauses and then applying the changes. Spinoza et al. (2009) have a similar full-automatic system under construction for Italian acts.

As MetaVex uses a different way to gather modifications, generating them from a modified version of the act rather than extracting them from the amending text, it may seem as if we already have a consolidated version: the version as modified by the user, which was used to generate the amending document. However, this only works if all changes are made in the right order. However, amending acts may be prepared in parallel rather than sequentially. In such cases, it is not possible to use the modified text resulting from the first modification as input for the second modification, as it possible that the second modification is being prepared at a time the first modification is also still being prepared. Moreover, it is unsure which modification will be applied first; it is even unsure whether they will get enacted. Also, modifications may be prepared in secrecy. Because the modifications are prepared in parallel, the final consolidation will not be one that has originally been created by one of the users, as neither of the two modified texts contains both change sets.

The consolidated version is generated by simply applying the changes to the previous consolidation. This can be done for a single amending document, but it also possible to apply several amending documents at the same time. This can help to detect conflicts between the different amending documents.

Conflicts between modifying acts can occur because they are prepared in parallel. For example, if one amendment introduces a new article dealing with the transport of bulk goods, while another amendment changes the definition of bulk goods, this may lead to unexpected results.

When a bill is pending at parliament, and its members propose amendments, conflicts are not usually a problem. These amendments are all prepared by the legislation bureau of the

parliament[179], where they are also checked for conflicts. When parliament votes on which amendments to accept, the voting procedure is arranged in such a manner that two conflicting amendments may not be accepted. However, amending laws can be prepared at different ministries, and at different times. This means there is less coordination, which may lead to clashing modifications being approved. Because of this, it is desirable to have support for the detection of these changes.

With an automated consolidation system, a certain group of clashes is easy to detect: situations in which one modification prevents another modification to be performed, e.g. an amendment repeals article 23 while another amendment wishes to modify article 23. If the automated consolidation system is used to apply these two amendments in this order, an error will occur. By trying out all different orders, such clashes can be detected. With this support for the generation of amending text and consolidations, as well as the basic editing functionalities, MetaVex is a solid, easily extendable and adaptable support for legislative drafters. It is still under construction, and may very well merge with similar tools being developed, such as the Norma Editor (Palmirani & Brighi, 2003) or xmlLeges Editor (Agnoloni, Francesoni & Spinosa, 2007), but we feel that eventually, an editor such as MetaVex will become the standard tool for legislative drafting.

6.3 Conclusion

Automated support for the creation of models of law is not the only product of the research presented in this book. As the examples in this chapter show, both the parsers developed as the knowledge gathered can be applied to other applications that help to make legislation more accessible.

[179] Bureau Wetgeving