Beyond the Experiment: the eXtendable Legal Link eXtractor

Marc van Opijnen
Publications Office of the Netherlands
marc.opijnen@koop.overheid.nl

Nico Verwer
Rakensi
nverwer@rakensi.com

Jan Meijer
Importalis
jmeijer@importalis.com

ABSTRACT

In this paper we describe a software framework for detecting and resolving references to (national and EU) legislation, case law, parliamentary documents and official gazettes. Meant to function in a large-scale production environment, performance, flexibility and maintainability are essential requirements. This led us to some noteworthy choices: within the pipeline architecture of Apache Cocoon we use the trie data structure for named entity recognition and a parsing expression grammar for pattern recognition, the latter having significant advantages over the use of regular expressions. Additional attention is paid to some substantive maintainability issues.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing – *language parsing and understanding*, *text analysis*.

General Terms

Algorithms, Management, Performance, Design, Reliability, Human Factors, Standardization, Languages.

Keywords

Legal semantic web. Natural Language Processing. Parsing expression grammar. Pipeline processing.

1. INTRODUCTION

The growing public availability of legal documents is a positive development. Under influence of the (amended) EU directive on Public Sector Information[1] and the G8 Open Data Charter[2] gradually more resources are becoming available as open data as well. However, while increasingly in structured and machine processable formats, the published information generally lacks machine processable links to other (legal) sources.

Within the Netherlands government it is recognized that the publication of essential legal sources like consolidated legislation, important judicial decisions and parliamentary documents in itself is not enough to cater for the information needs of both the public sector itself and the citizens. Interlinking legal sources was considered to be an important prerequisite for improving governmental efficiency and effective knowledge management. Hence, a project for 'Linked Governmental Data' (Dutch abbreviation: 'LiDO') was initiated: based on semantic web technologies repositories from various governmental institutions are collected, with the goal of providing insight into the relations between legislation and other public legal documents.

Although the number of documents collected is starting to be impressive, the available links are mainly those already available in the formal metadata. The number of additional – editorial – links is quite limited, while exactly those are having great additional value. Just making explicit all those textual links that are currently not computer readable would give real added value. Since human tagging of the more than a million documents available is far too costly, automated extraction of references is the only viable alternative.

Quite some research has already been done on legal reference parsing, most of it in an academic or experimental setting. One of the most elaborated projects on linking Dutch legal data has been realised within the framework of the research on a Model for Automated Rating of Case Law [1, 2]. Although the functionalities built within this research project were quite extensive, the software was developed with the purpose of populating a research database just once, and not to serve as a production environment. Performance could clearly be improved, the software was not maintainable, and was not designed for a linked data environment. Our mission therefore was to completely redesign and rebuild the technical framework of this link extractor and to extend its capabilities, while at the same time reusing its intelligence for the proper recognition of legal references.

We will start with listing the most important requirements for this redesign (§ 2). In § 3 we will discuss the main components. After explaining the choice of framework (§ 3.1) specific attention will be paid to the choice of the trie data structure for named entity recognition (§ 3.2) and a parsing expression grammar – instead of regular expressions – for pattern recognition (§ 3.4). In § 4 some challenges are discussed which are of particular relevance when deploying a link extractor in a production environment, like the use of external reference repositories and changing standards for identification and citing of legal sources – illustrated with some examples from the European domain.. In § 5 some conclusions are drawn.

Our eXtendable Legal Link eXtractor (hereinafter also referred to as 'xLLx') was designed for use on Dutch texts, recognizing links to both national and EU sources. It could be used on other languages as well though. All code examples in this paper have been translated into English.

2. BASIC REQUIREMENTS

Our requirements can be grouped under four headings: a flexible and extendable architecture, stability and performance, maintainability, and finally a broad variety of links to be detected.

The flexibility needed in the architecture relates to several aspects. Two important tasks are the recognition of named entities and pattern matching. Specific functions like normalization, disambiguation, canonicalization should be easy to integrate.

[1] Directive 2013/37/EU of the European Parliament and of the Council of 26 June 2013 amending Directive 2003/98/EC on the re-use of public sector information (CELEX: 32013L0037)

[2] https://www.gov.uk/government/publications/open-data-charter/g8-open-data-charter-and-technical-annex.

Since we depend on both internal and external reference repositories for named entities, metadata and URIs, a seamless integration of, and connection to these repositories is essential. Of course xLLx must be made available as a service, and the API must cater for a variety of user needs, like defining the type of links to be recognized (e.g. just to legislation, case law or parliamentary documents), or to define instructions which are relevant for specific documents only. And pre-processing and post-processing should of course be easily configurable.

Regarding performance it should be mentioned that the number of named entities to be recognized is around 200.000, while the number of patterns to be recognized might be quite limitless. Of course performance is dependent upon hardware, but as a rule of thumb the hardware requirements should not be substantially influenced by the complexity of the patterns or the number of named entities to be recognized. Proven technology and a scalable architecture are also obvious requirements.

The third requirement concerns the maintainability of the software, more specifically the functional part of it. Apart from general rules regarding the use of non-proprietary software, the availability of unit tests, limiting the lines of code and proper documentation, natural language processing software that has to recognize extremely complex strings – like legal references – runs the risk of become overly complex by a lack of modularity and long and hence unstable pattern matching code. As some examples in § 4.2 will demonstrate, citation instructions and habits change, as do reference repositories and the structure of input documents. These changes require domain specific knowledge, and making changes and adding new functionalities must be doable even after the architect and the main developers have left the scene.

The final requirement is of a functional nature: it concerns the types of links to be detected and resolved. In [1, 2] links to (national and EU) legislation and case law were detected, xLLx should also be able to detect references to parliamentary documents and publications in official gazettes, of which there are seven different types in the Netherlands.

3. MAIN COMPONENTS

3.1 Pipeline Architecture

For natural language processing various frameworks are in use, also within the legal domain [3]. The most important of these platforms have been reviewed by us, but for various reasons they were not considered fit, especially since they lack the flexibility and adaptability needed for the very specific task of legal reference parsing.

A first choice was to use Apache Cocoon, an XML processing framework based on the concepts of pipelines, separation of concerns and components. We already had Cocoon running in our linked data architecture, and this was therefore considered with priority. It's simplicity of concept but at the same time enormous flexibility and extendibility made it a sensible choice.

Within Cocoon a 'pipeline' can be defined as a series of linked 'components' that can generate, transform and serialize XML content. A 'sitemap' specifies the pipelines for a specific type of request. As a result, components can be used within different pipelines. The Cocoon pipeline architecture is extremely flexible, making it possible to apply all kinds of logic and (XSLT) transformations.

Within our linked data framework xLLx is configured as a service, and makes use of other RESTful services, mainly to connect to reference repositories. These repositories are built outside xLLx, but are using the same Cocoon architecture for collecting, transforming and storing content from internal and remote sources.

In this paper we cannot discuss the whole pipeline architecture and all of its components. We limit ourselves to describing the main components in the pipeline for detecting and resolving references to national legislation, many of which are also used in other pipelines – therefore on some occasions examples from other pipelines are used for illustration. In § 3.6.3 a remarkable component from another pipeline is discussed.

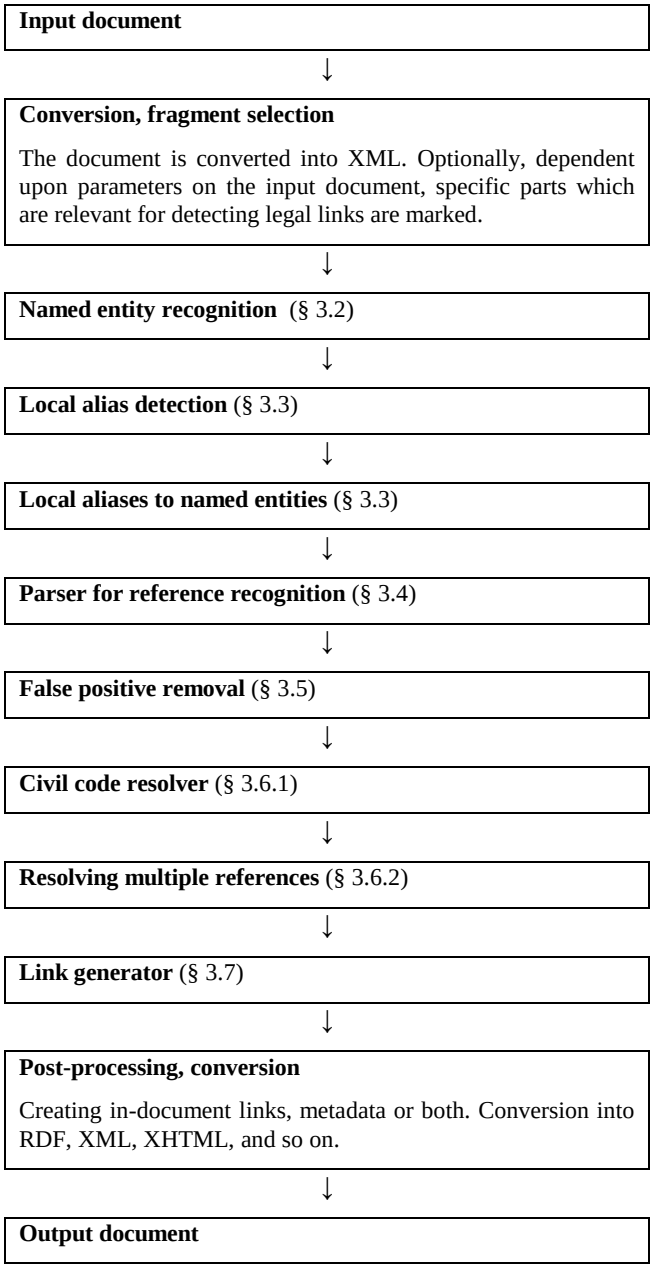

Figure 1. Schematic display of pipeline for detecting and resolving references to national legislation.

Figure 1 shows a schematic display of the pipeline. If a component needs detailed explanation there is a reference to the relevant paragraph.

One could observe there is no component for human editing. Since xLLx is meant for completely automated high volume processing a component for human editing is not essential for the pipeline. If needed it can be added easily. Moreover, it should be noted that many components are configured to help a human editor by throwing 'resolve manual' errors: they indicate that a link probably has been detected but that the exact reference cannot be established.

3.2 Named Entity Recognition

3.2.1 Introduction

Named entity recognition (NER) is one of the hardest problems to tackle in natural language processing [5]. Deviating a little from the general theory and tailored to the legal domain, we distinguish between 'named entities' and 'patterns'. Named entities are single names that identify a legal resource, like official and non-official titles (including abbreviations). Patterns are strings that (might) be a reference because of their specific structure.

The relevance of being able to make a distinction between the two can be illustrated by the following example. An author might make a reference using the title of an EU directive, reading: *"Commission Directive 2014/110/EU of 17 December 2014 amending Directive 2004/33/EC as regards temporary deferral criteria for donors of allogeneic blood donations"*. If a parser is only trained to detect patterns of EU directives, it will recognize both 'Directive 2014/110/EU' and 'Directive 2004/33/EC'. Linking to both these directives though was not intended by the author; he only wanted to refer to the first one, contrary to a situation in which he writes: *"The substantive provisions of Directive 2004/33/EC were not affected by its amendment by Directive 2014/110/EU."*

From this example it follows that we should first detect named entities, and subsequently look for patterns. Hence, a pattern that is a part of a named entity is not recognized as a pattern but only as (part of) the named entity. We have to be aware though that it can also be the other way around: a named entity being part of a pattern. The abbreviation 'RVS' stands for 'Law on the Council of State' and as such is a named entity. But 'RVS' can also be part of a European Case Law Identifier (ECLI),[3] identifying a judicial decision of the Council, as in: ECLI:NL:RVS:2015:985. In such a situation the pattern should take precedence over the named entity.

Given the fact that the number of EU legal acts is well over a hundred thousand, and for national legislation it is in the tens of thousands, it is obvious that machine learning (e.g. by fuzzy matching) might easily lead to a disappointing precision.

Because on citing full titles – like in our first example – are often literally copied from a reference source, NER can be used to detect them, but when shorthand notations are used or references to e.g. paragraphs of law are made, we have to use pattern recognition instead. To stick to the example: chances are the author doesn't cite with 'Directive 2004/33/EC', but by 'Dir. 2004/33', 'EC directive 2004-33' or one of many other possible variants – unfortunately the creativity of lawyers in circumventing citation guidelines is sheer unbounded.

3.2.2 Trie Data structure

Many packages for entity recognition are based on data structures like trees, arrays or linked lists. These work well if the strings to be recognized are relatively small in length and volume. Our set though is quite voluminous: it contains already more than 200.000 strings: all titles of secondary EU legislation (the sectors 2, 3 and 4 from EUR-Lex), all titles from the consolidated legislation database of the Netherlands, and some thousands of commonly used aliases for both European and national legislation and case law. Apart from the number of strings to be recognized, the strings are quite lengthy, as the example of the EU Commission directive above shows.

After having tried various parsers, we ended up with a trie-implementation. Trie is a somewhat neglected or misunderstood data structure that is well fit to solve problems like ours [6]. It is extremely powerful: its performance only depends on the length of the prefix substring shared by at least two entities and not on the size of the vocabulary.

For optimal performance the whole (multidimensional) tree has to be loaded in memory, but because of the compression inherent in a trie this does not require exceptional hardware.

The package used needed some adjustments for natural language processing: it had to be made UTF-8 compliant to replace diacritics by their non-diacritic version and expand ligatures and digraphs; using only ASCII and removing all whitespace and punctuation marks made NER immune to common variations in spelling and further sped up performance. It had to be wrapped to function in a Cocoon pipeline and from a functional point of view it had to be extended to return not the string matched, but the URI that belongs to that string (e.g. the CELEX number for a European directive). And while trie normally works either case-sensitive or case-insensitive, we had to tune it to work case-sensitive for strings up to (in our case) six characters, and case-insensitive for longer strings. The reason for this being that casing must be ignored in long titles since casing errors are often made, while for (short) abbreviations there are differences in meaning between uppercase, lowercase and mixed case variations, while there is also the risk of the abbreviation coinciding, though in another casing, with common words.

By default trie takes the longest possible match, but needless to say that's exactly the functionality we want. When we parse a text referring to the judicial decision 'Rensing/Polak II', trie should not match it on the case 'Rensing/Polak I', but only on 'Rensing/Polak II'.

For populating the data tree we use the already mentioned sources. Caution is needed though: they contain names that are not only titles of legislation, but might also quite commonly appear as common words. As an example from the EU domain CELEX:32015D0213(01) could serve: it has as its title (just) "Decision" (as many other records in EUR-Lex). Since this would lead to false positives in any situation where the word 'Decision' is used in a processed text, we created an 'exception list': entries from internal and external resources that have to be prevented from being included in the trie dataset. Although one could consider more sophisticated solutions, the best alternative for now is to have the delta in the reference repositories checked by an editor before they are inserted into the collection.

[3] *Infra*,§ 3.6.3.

3.3 Local Aliases

As already touched upon in § 3.2 lawyers often use aliases for referring to well-known laws or judicial decisions. Since these aliases are shared within the legal community we call them 'global aliases'. They can be recognized as named entities. Apart from these global aliases also local aliases are used. Like variables in software code they are declared on first use, and they don't have any value outside the document in which they are declared. Declaration often takes the form: *"the European Convention on Human Right and Fundamental Freedoms (hereinafter 'the Convention')."* When two pages further a reference is made to "article 6 of the Convention", 'the Convention' – the local alias – is immediately understood by the human reader.

To achieve the same understanding of the text two components were introduced in xLLx. The component 'local alias detection' searches for local aliases after each named entity found, and the component 'local aliases to named entities' detects other occurrences of the local aliases found and tags them accordingly.

3.4 Pattern Recognition: Parsing Expression Grammar

Legal references not using the official or non-official title of a legal document have very specific formats. Although citation guides exist and are often followed, deviations from these semi-formal guidelines are the rule rather than the exception. This can be illustrated by the analysis of Dutch, German and British judicial decisions, that yielded tens of different formats for citing EU regulations and directives [7].

For a link extractor to function properly all these variations have to be captured. Regular expressions ('regexes') are a commonly used technique to tackle this problem, e.g. used in [4, 7, 8, 9, 10].

Although they are well fit for recognizing credit card numbers or dates, regexes have some serious drawbacks when it comes to recognizing legal references in a large-scale environment.

Firstly, regular expressions nowadays often extend the original regex formalism with features like look-aheads and greedy or lazy repetitions. These extensions not only lack a formal standard and therefore have different implementations in various programming languages, but they are difficult – i.e. time-consuming – to develop and even harder to maintain. For time- and scope-limited academic research – like most of the works cited above, where only a limited set of references is to be detected or a limited number of documents processed – this is no serious objection, but when implemented in a production environment ten or twenty lines of regexes become extremely hard to maintain.

Secondly, in complex implementations, the use of long regexes can seriously slow down performance, especially when using functionalities like backtracking.[11]

The third reason is of a more fundamental nature: regexes cannot solve the problem of ambiguity easily. But when it comes to legal references we want to exclude ambiguity as much as possible. As an illustration we use the way Dutch parliamentary documents are identified and can be cited.

Although in reality it's a little more complex, for our purposes we can simplify it as follows. Every document identifier starts with a dossier code (identifying e.g. the set of documents eferring to a specific legislative proposal), written in five digits. Optionally it can have a dossier subcode attached – used mainly for budget proposals – separated from the dossier code by a hyphen and written in Roman numerals, but other capitals exits. Finally it has a sequence number in digits, indicating the serial number within the dossier, preceded by 'nr'. Sometimes though the sequence number is written in capital letters.

So, ideally a document number looks likes:

- 12345 nr 6
- 12345 nr AA
- 12345-IV nr 5

We can capture this properly with a regex like:

```
[1-9][0-9]{4} \s nr \s [A-Z]{1,5}
|
[1-9][0-9]{4} (-[A-Z]{1,5})? \s nr \s [1-9][0-9]?
```

The problems start with poor citations: often 'nr' is left out or the hyphen between the dossier code and the dossier subcode is replaced by whitespace or just left out.

To cater for these possibilities we have to re-write our regex, e.g.:

```
[1-9][0-9]{4} ( \s (nr \s)? [A-Z]{1,5} |
 ((-|\s)?[A-Z]{1,5})? \s (nr \s)? [1-9][0-9]? )
```

Although many citations are recognized correctly, we have a problem with: '12345 IV nr 5'. Only '12345 IV' will be detected, which is not an existing document number. The solution would be not to allow the whitespace between dossier code and the subcode, but then we wouldn't recognize 12345 AA, a notation that is commonly used in case 'AA' is the sequence number.

We came across many of such problematic ambiguities. In theory, these problems could be solved with regular expressions, using many repeated parts and many nested alternatives. The result quickly becomes unwieldy and prone to subtle errors. In conjunction with the first two drawbacks of using regexes we decided to exchange regexes for a parsing expression grammar.

A parsing expression grammar (PEG) is a formalized machine-oriented syntax introduced by Bryan Ford in 2004 [12]. It differs from context-free grammars (CFGs) in its ability to eliminate ambiguity. Instead of an unordered choice operator ('|') as used by CFGs, a PEG uses a prioritized choice: alternative patterns are tested in order. Generally, because of their ability to cope with ambiguity, CFGs are often considered to be better suited for natural language processing [12] than PEGs, but for references from the legal domain the non-ambiguity is an advantage rather than a drawback, and its – implicit – longest match recognition capability makes a PEG the better choice.

Using a formalized grammar like PEG offers more advantages, like modularization of patterns in 'non-terminals' by which parse-trees are constructed. As a result development is faster and less error-prone. The code is more human readable and can be documented better, which improves maintainability. Because of the abstract syntax tree (AST) in which the grammar returns its results, all recognized strings can be reused easily in the next step of a pipeline.

In the following we demonstrate how a PEG can solve the abovementioned problem, at the same time illustrating the charm of modularization.

First some PEG basics. A grammar consists of a set of rule definitions, the already mentioned 'non-terminals'. They specify in a formalized way (small) strings of text that have to be

recognized. Non-terminals are defined in three parts: a name, a rule type and some grammar expressions.

The non-terminal for the parliamentary dossier code from our earlier example is of the most common rule type, the 'tree constructing type' (written as '<-'). We name it 'DosCode'. It reads:

```
DosCode
        <-
        [1-9] [0-9] [0-9] [0-9] [0-9]
```

In our example we only use this tree constructing type; so we will not discuss other types. The grammar expressions look very much like regular expressions, with two important syntactic differences: modifiers (like '?' for optionality) are written before and not after the grammar expression, and literal strings are within single (case-sensitive) or double (case-insensitive) quotes. A non-quoted string is a reference to another non-terminal.

Given the citation guidelines and the common aberrations, we can construct some other basic non-terminals for parliamentary documents. The non-terminal 'sp' stands for whitespace.

For the dossier subcode:

```
DosSubCode
        <-
        [A-Z] ?[A-Z] ?[A-Z] ?[A-Z] ?[A-Z]
```

For the separation between the dossier code and the dossier subcode we can write the following (note that only the optionality of the comma is defined here, not the optionality of the separator as such):

```
DosSeparator
        <-
        ?',' sp | '-'
```

To define the serial number:

```
DocSerial
        <-
        [1-9] ?[0-9]
        |
        [A-Z] ?[A-Z] ?[A-Z] ?[A-Z] ?[A-Z]
```

And finally, to define the label for the serial number:

```
SerialLabel
        <-
        ("nr" sp)
```

Now we can use these non-terminals to construct a non-terminal for the whole reference:

```
ParliamentaryDoc
        <-
        DosCode ?(?DosSeparator DosSubCode) sp ?SerialLabel
        DocSerial
```

Although this code already demonstrates the advantages of a PEG regarding modularity, human readability and flexibility, we still have the same problems as with the regex: '12345 IV nr 5'will be recognized, but '12345 AA' will be understood as a dossier (where 'AA' is the dossier subcode) and not as a dossier code + document number.

To solve this we have to utilize the prioritized choice, fundamental to a PEG:

```
        DosCode ?DosSeparator DosSubcode sp ?SerialLabel
        DocSerial
        |
        DosCode sp ?SerialLabel DocSerial
```

The prioritized choice means that as soon as an alternative gives a match, it doesn't look for other alternatives. For that reason we have to place the longest possible match on the first position. If we parse '12345 IV nr 5' it will be tested against the first alternative, with all elements matched to the correct non-terminals. '12345 AA' doesn't match the first alternative, since it would need both a dossier subcode and a document number. It is matched against the second alternative though, with 'AA' correctly identified as the document sequence number.

For implementing a PEG we choose the Waxeye parser generator, which could be integrated seamlessly in Cocoon. PEG is not only used to detect main entities – like references to European directives – but also to detect more granular references, connected to main entities – whether being detected as named entity or as a pattern – e.g. to chapters, articles and paragraphs. Altogether, more than 1700 lines of grammar (including comments) were written for the recognition of various types of references and for canonicalization (see § 3.6.3). The result has proven to be very readable and maintainable, even by non-programmers.

3.5 False Positive Removal

False positives occur, most commonly when an abbreviation of a law or regulation equals something completely different in the document processed, e.g. a broadcasting company, the initials of a barrister or the code of a referenced case file. Without context analysis the difference is hard to tell, but the occurrence of such false positives can be drastically reduced by removing all string matches of up to six letters (being abbreviations) if they are not accompanied by a more granular reference, e.g. to a specific article or paragraph. An exception to the latter restriction is made for situations where such an abbreviation is defined as a local alias: we can assume that the author of the parsed document wouldn't introduce ambiguity himself.

3.6 Content Specific Components

Although not all pipelines use all components described in the previous paragraphs, they are of a general nature. In some pipelines very specific components are used, tailored to particular needs. We will not list all of them, but just give some examples. In § 3.6.3 we describe a specific component for resolving references to case law, but first we discuss two specific components from the legislative references pipeline: the civil code resolver (§ 3.6.1) and the resolver for multiple references (§ 3.6.2).

3.6.1 Civil Code Resolver

The Dutch Civil Code contains ten 'books'. In the legislation database all these books have a separate identifier, and consequently they would only be recognized in the NER component by their titles: 'Civil Code Book 1' and so on. Unfortunately though, if a lawyer makes a reference to article 269 of this first book, he usually writes: "Art. 1:269 Civil Code" or something comparable: the book identifier is part of the article

reference and not of the main entity. To solve this issue the NER component does recognize 'Civil Code' as such, but does not return an identifier, since the title is ambiguous. In this component we disambiguate the title – by using the first digit in the article number, or resolving other styles used to indicate the book – and rewrite the article reference to '269' to construct a valid URI. In some references the specific book isn't mentioned – erroneously or because the author assumes the reader understands it within the given context – in which case the component throws a 'resolve manual'.

3.6.2 Resolving Multiple References

This component resolves multiple references, like in: "Articles 2, 3 and 9 to 13 of regulation XYZ". Five visual links are created – the four articles mentioned and the regulation itself – but also three extra hidden links – to the articles 10, 11 and 12. Improvements though are still possible: currently there is no check against the repository on the specific articles in the range mentioned. So, if an article 11a exists, we would miss it.

Next to such sibling references this component also resolves hierarchical references like: 'Part 3 of chapter 3'. The nesting is relevant, since also chapter 6 might contain a 'part 3'. The information from this component is used for the link generation described in § 3.7.

3.6.3 Canonicalizing Complex Case Law Citations

As demonstrated in the previous paragraphs, references to legislation can be quite complex, but apart from the mostly well-defined occurrence of local aliases, citations are singular in the sense that only one entity is used to refer to a source.

As in many other jurisdictions, in the Netherlands case law references are much more complex: a judicial decision can be cited by one or more references to nearly a hundred different case law periodicals, the (rather form-free) case number (in combination with the name of the court and the date of judgment), the national case law identifier ('LJN') which in June 2013 is replaced by the European Case Law Identifier (ECLI),[4] or a combination of some or all of them. The 'canonicalization' process needed to solve this puzzle has been described extensively in [8]. To summarize: all elements that could be part of case law citation are detected, normalized and canonicalized to an overarching identifier by using a register that contains all case numbers, parallel citations and case law identifiers (LJN and ECLI).

Reusing the algorithms of [8] the code was completely redesigned and some functionality added. Apart from the recognition of ECLI and using it as the canonicalized identifier (instead of its national predecessor), we also added the recognition of the often cited opinion of the advocate-general, which functions both at the Dutch Supreme Court as well as at the Court of Justice of the European Union (CoJ).

This can be illustrated by the following text fragment, in which the links are constructed by xLLx:

> *We should compare this to the theory described in the opinion of Kokott in Case C-231/05, Court of Justice, 18 July 2007 (Reports 2007 I-06373).*

The first link leads to the opinion (ECLI:EU:C:2006:551), the second to the judgment (ECLI:EU:C:2007:439). Strictly speaking the reference to the judgment itself is – in this specific sentence – a false positive, since the author of this text only referred to the judgment to identify the opinion.

Also the 'Reports' number used in this citation is noteworthy. Apart from the fact that many spelling variants are used by authors and hence it is hard to detect properly, it is in itself ambiguous because it identifies both opinion and judgment, which – as shown – both have their own ECLI. The xLLx canonicalization component defaults to the judgment; if the opinion is meant also or instead, it will (also) be recognized, as shown in the example.

3.7 Link Generation

In the 'link generation component' URIs are constructed for the references found. For case law it uses ECLI, for parliamentary documents and official gazettes the URIs catered for by the official repositories. For European references the CELEX number is used. For national legislative references 'Juriconnect-BWB' is used: a national URI standard for referencing (elements within) legislation. Juriconnect-BWB uses the identifier of the national database for consolidated legislation and can be fully resolved by (at least) that database. It has a strict format for referring to articles and other elements. It can also contain two dates: one for the validity and one for the date of viewing (relevant for retroactive changes). Default the constructed links are without date of viewing; as the date of validity we take the production date of the document in which the reference is detected. For specific pipelines these choices can of course easily be altered.

As follows from this description, all links are basic URIs, no HTTP-URI's. Specific resolvers have to be configured by the applications using the output documents.

4. MAINTENANCE ISSUES

In an experimental or academic setting one can use a limited set of reference data [4], or extract links for populating a research database just once [1], without having to worry about typical maintenance or managerial issues.[5] In this paragraph we discuss three of them: connecting to external sources, changing identifiers, metadata and citation habits and functional maintenance.

4.1 Reference Repositories

For proper recognition and resolution we currently use two internal databases from the Dutch Publications Office and two external repositories. The internal databases contain the consolidated legislation, the (currently seven types of) official gazettes and parliamentary documents. For national case law we use the open data of the Council for the Judiciary and for European case law and legislation we use the EUR-Lex webservice.

[4] Council conclusions inviting the introduction of the European Case Law Identifier (ECLI) and a minimum set of uniform metadata for case law (CELEX:52011XG0429(01)). See also [13].

[5] LawCite is an example of a large-scale link extractor in a production environment, but although a paper has been written about it [14], the architecture or managerial issues are not publicly described.

Anyone using external APIs knows that they all have their own reliability issues, might change unexpectedly and are often poorly documented. Strict monitoring is required and creativity needed. As an example[6] might serve the problems to build a proper reference repositories of EU ECLI's. On the introduction of ECLI by the Court of Justice[7] in 2014 ECLIs assigned to all decisions and opinions of the CoJ. They were displayed on the Court's website and they were also supplied to the EUR-Lex database. But while the court continued to display newly assigned ECLIs on its website, ECLIs assigned after July 2014 are not visible within EUR-Lex. They have to be harvested from the Court's website instead, which is more cumbersome since that website does not have an API.

4.2 Changing Standards

Another example from the European domain can be used to demonstrate the impact of the ever ongoing changes in identification systems and citation rules. The change is futile for lawyers citing European acts, but it has quite an impact on software like ours.

On January 1st 2015 the formatting of European legal acts changed, in two aspects.[8] The first issue concerns the change from parallel to non-parallel serial numbering for different types of legal acts. Before 2015 directives, regulations and decisions all had their own sequence numbering: in any given year there could be a number '10' for all types of acts. From 2015 onwards they all share the sequence, so in any given year there can only be one act carrying number '10'. This change does not impact xLLx, but the second one does: the numbering formats have been harmonized. Previously there were divergent practices for the formatting of the various types of legal acts.

The identifier of a directive or decision used to be formatted as:

[type] [year]/[number]/[domain]

e.g.: Directive 2003/98/EC

For a regulation the proper format used to be:

[type] ([domain]) No [number]/[year]

e.g.: Regulation (EEC) No 1408/71

In the new situation all types share the same formatting:

[type] ([domain]) [year]/[number]

e.g.: Regulation (EU) 2015/539

Because of the first change the type is not needed anymore to disambiguate. The domain (e.g. 'EU', 'CFSP') has never been needed for this purpose at all. Whether or not they currently have to be part of a proper citation is not completely clear from the explanation of the Publications Office:[9] in the main rule only the domain is expressed, but in the examples the type is also mentioned. In citation practice though we can expect the continued mentioning of the type, with the domain often left out of the reference.

Aware of the liberal citation habits of the average lawyer, the xLLx grammar already took into account the many different spelling variants for European legal acts. Completely misformatted references like 'Dir. 2003-1998' and 'EC-Regulation 1408.71' are recognized without any problem. [7]

For directives and decisions the changed format was no problem: the domain was already an optional element in our grammar, its position could vary and mixing up 'EEC', 'EC' and 'EU' was taken into account.

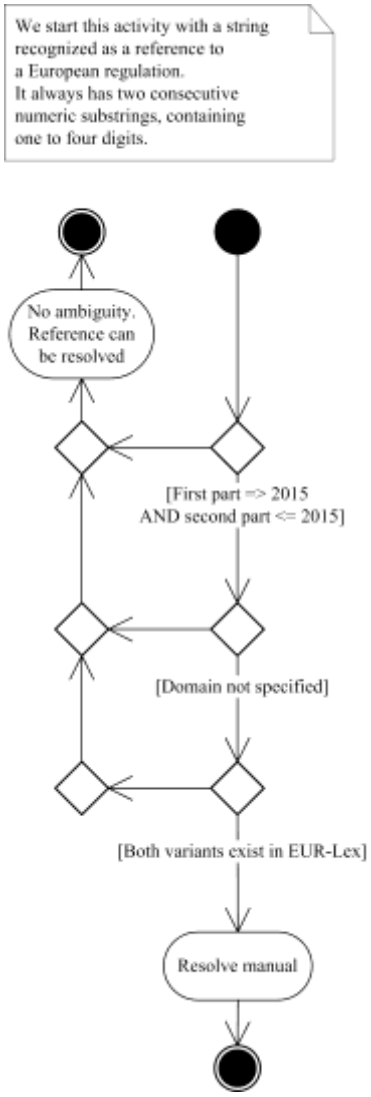

Figure 2. Activity diagram for establishing correct CELEX number for reference to European regulation.

The change in the numbering of regulations though has a more serious impact, because the year-number sequence changed, it used to be 'number/year', from 2015 onwards it is 'year/number'. In our grammar also for regulations the domain is an optional element, and except for years from the new century, the year might be written in two or four digits,[10] otherwise precision would

[6] At the time of writing (02-04-2015).

[7] http://curia.europa.eu/jcms/jcms/P_125997/

[8] http://eur-lex.europa.eu/content/tools/elaw/OA0614022END.pdf

[9] See footnote 8.

[10] Although according to the Interinstitutional style guide years up until 1998 have to be written in two digits, and from (and including) 1999 in four digits. . (http://publications.europa.eu/code/en/en-110302.htm)

be too low. As a result 'Reg. 1408/1971' will be recognized properly, as will 'Regulation EC 93/92'.

Because of the harmonization we had to devise a new grammar for regulations. This new grammar for 2015 and onwards requires '2015' (or higher) on the first position, but this poses a serious problem for a citation like 'Reg. 2015/75': it can refer to Regulation 75 from the year 2015 (CELEX:32015R0075), or regulation 2015 from the year 1975 (CELEX:31975R2015) – and they actually both do exist.

To improve the correct recognition we became more strict on the difference in domain between EEC/EC on the one hand and EU on the other, and introduced additional logic to determine which variant probably was meant by the author.

Implemented in a co-operation between grammar and logic, the rule now goes as illustrated in the activity diagram of Figure 2.

A European development we have not been able to implement yet is the European Legislation Identifier (ELI),[11] since it has not yet been introduced on EUR-Lex. We are eagerly awaiting this introduction, especially if it caters for granular references to EU legal acts. Currently, by using the CELEX number, one can only refer to a legal act as such, not to a specific article within the act. ELI should make this possible.

Like with national legislation, in xLLx we do recognize article references in conjunction with the European instruments themselves, so when EUR-Lex introduces ELI URIs to elements within acts, we would be fully connected by modifying just one XSLT stylesheet.

4.3 Functional Maintenance

As might have become apparent already, a link extractor is maintenance sensitive. Apart from monitoring developments regarding external resources and standardization efforts as described in the previous paragraphs, there are also functional maintenance tasks. Checking and possibly improving the links found – e.g. to process the 'resolve manual errors' – is a task for end-users, but it's a responsibility of the system manager to keep it functioning. This entails of course technical maintenance, configuring grammars and sometimes pipelines for new document types, but also editorial work, e.g. to have the exceptions list (§ 3.2.2) updated.

5. CONCLUSIONS

In this paper we have described the architecture of a legal link extractor that is flexible, extendable and deployable in a production environment. The legal domain is very specific and needs specific solutions, for which we hope to have catered. On finishing this paper the last bugs were just fixed, and it is not live yet. However, we expect it to be in production within a couple of months.

With some specific measures like caching of tries and compiled grammars, we were able to make the xLLx perform very well. In the most extended pipeline (recognizing all types of references), a typical case law document is processed within 2 to 10 seconds on very moderate hardware, time spent in communicating with external repositories included.

[11] Council conclusions inviting the introduction of the European Legislation Identifier (CELEX:52012XG1026(01)).

For testing purposes we have randomly selected two judicial decisions from each of the seven types of courts in the Netherlands. The results for recall and precision are displayed in Table 1.

Table 1. Recall and precision of xLLx.

Reference to:	Relevant	Detected	False positive	Recall	Precision
National legislation	271	265	10	98%	96%
EU acts	69	64	1	93%	98%
Elements of (national and EU) legislation	204	183	0	90%	100%
Case law	151	140	1	93%	99%
Parliamentary documents	15	13	0	87%	100%
Official journals	9	9	0	100%	100%
Total	719	674	12	94%	98%

Of course there is always room for improvement. Some possible functional extensions have already been mentioned, and we came across several other options for more advanced fine-tuning. But as long as legal writers are not legally obliged to cite properly and/or they are not supported sufficiently by intelligent authoring tools, there will always be unresolvable references made. But on achieving this performance, we hope to have attained our goal of substantially improving the accessibility of legal documents.

6. REFERENCES

[1] M. van Opijnen, Op en in het web. Hoe de toegankelijkheid van rechterlijke uitspraken kan worden verbeterd, PhD University of Amsterdam, 2014.

[2] M. van Opijnen, A Model for Automated Rating of Case Law, Fourteenth International Conference on Artificial Intelligence and Law, ACM, New York, 2013, pp. 140-149.

[3] G. Boella and H. Kostantinov, Report on the state-of-the-art and user needs, EU Cases, 2014. http://eucases.eu/d1_1/

[4] E. de Maat, Making Sense of Legal Texts, PhD University of Amsterdam, 2012.

[5] S. S. David Nadeau, A survey of named entity recognition and classification, Lingvisticæ Investigationes 30:1 (2007): 3-26.

[6] A. C. Bellini, The Trie: A Neglected Data Structure, www.toptal.com/java/the-trie-a-neglected-data-structure.

[7] M. van Opijnen, Searching for References to Secondary EU Legislation, in S. Tojo, ed., Fourth International Workshop on Juris-informatics (JURISIN 2010).

[8] M. van Opijnen, Canonicalizing Complex Case Law Citations, in R. Winkels, ed., Legal Knowledge and Information Systems - JURIX 2010: The Twenty-Third

Annual Conference, IOS Press, Amsterdam, 2010, pp. 97-106.

[9] L. Bacci, E. Francesconi and M. Teresa, A Proposal for Introducing the ECLI Standard in the Italian Judicial Documentary System, Legal Knowledge and Information Systems - JURIX 2013: The Twenty-Sixth Annual Conference, IOS Press, 2013, pp. 49-58.

[10] R. Winkels, J. de Ruyter and H. Kroese, Determining Authority of Dutch Case Law, in K. M. Atkinson, ed., Legal Knowledge and Information Systems. JURIX 2011: The Twenty-Fourth International Conference., IOS Press, Amsterdam, 2011, pp. 103-112.

[11] R. Ierusalimschy, A Text Pattern-Matching Tool based on Parsing Expression Grammars, Software: Practice and Experience 2008).

[12] B. Ford, Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, New York, 2004, pp. 111-122.

[13] M. van Opijnen, European Case Law Identifier: indispensable asset for legal information retrieval, in M. A. Biasiotti and S. Faro, eds., From Information to Knowledge. Online Access to Legal Information: Methodologies, Trends and Perspectives, IOS Press, Amsterdam, 2011, pp. 91-104.

[14] A. Mowbray, P. Chung and G. Greenleaf, Free-access Case Law Enhancements for Australian Law, in G. Peruginelli and M. Ragona, eds., Law via the Internet. Free Access, Quality of Information, Effectiveness of Rights, European Press Academic Publishing, Florence, 2009, pp. 285-298.

This work is part of the 2015 Workshop on Automated Detection, Extraction and Analysis of Semantic Information in Legal Texts, held in conjunction with the 2015 International Conference on Artificial Intelligence and Law (ICAIL), June 08 - 12, 2015, San Diego, CA, USA.