Tom M. van Engers, Ron van Gog and Kamal Sayah, 'A Case Study on Automated Norm Extraction' in T. Gordon (ed.), *Legal Knowledge and Information Systems. Jurix 2004: The Seventeenth Annual Conference*. Amsterdam: IOS Press, 2004, pp. 49-58.

A Case Study on Automated Norm Extraction

Tom M. van Engers[1], Ron van Gog[2], Kamal Sayah[3]
[1] *Leibniz Center for Law, University of Amsterdam,* T.M.vanEngers@uva.nl
[2] *Dutch Tax and Customs Administration/CPP,* rvangog@planet.nl
[3] *University of Utrecht,* ksayah@cs.uu.nl

Abstract. Within the (E-)Power research program a new approach for supporting the chain of processes from the creation of legal texts to the implementation of normative (juridical) information systems has been developed. According to this approach creating formal knowledge models starts with the analysis of the legal text. This process executed by knowledge analysts is very time consuming. Within the Power program an automated concept and norm extraction tool and a model generation tool using linguistic techniques have been developed to improve modelling productivity. This paper describes how this tool can be used to translate legal texts into a formal model (in our case the UML/OCL standard). We will do this by giving some examples of specific legal constructs and discuss the modelling process.

1 Introduction

In 1990 the Dutch Tax and Customs Administration (DTCA) started a research programme aimed at improving the whole chain of processes starting with legal drafting to the design and implementation of business processes and ICT solutions supporting the operationalisation of legislation in the operational units of the DTCA. The core of this programme called POWER (Programme for an Ontology-based Working Environment for design and implementation of Rules and regulations) was the knowledge representation of normative knowledge (i.e. knowledge what is obliged, prohibited or permitted) and the extraction of this knowledge from the different legal knowledge sources such as the law, directives, case law decisions and of course legal expertise extracted from human experts.

One of the aims of the POWER-programme besides being able to improve legal quality and improve accessibility of the law was to reduce the 'time to market', i.e. the time needed for implementing new regulations in the administrations operational units and providing both civil servants and the administrations' clients with adequate support. Another aims was the reduction of maintenance costs, since about 80 percent of the administrations' investments in ICT solution, in the case of the DTCA about 500 million euro on an annual basis, is spend on maintenance.

A more systematic way of extracting knowledge from the legal knowledge sources and a standardised way of formally expressing that knowledge would certainly improve the existing situation. During the first years of the POWER-programme the knowledge analysts involved in developing the POWER-approach developed a method that proved to result in applicable knowledge representations. After analysing their way of working we discovered that the syntax of the natural language expressions in the law texts provides all the handles we needed for automation of the concept extraction (see [3]). We developed an automated support system (part of the Power-workbench) that proved to be helpful in several projects. The automated norm extraction tool not only saved the knowledge analysts a lot of time, but it also contributed to more uniform knowledge representations (less inter-coder dependency) and consequently improved their maintainability.

A more ambitious research question was if we would be able to also create automated support for extracting the norms from the law texts (and other legal knowledge sources) using similar natural language processing techniques (NLP). After two years of research in this field we now are confident that we can reach a significant level of support by automated norm extraction. Thus far we succeeded in implementing parts of that functionality in a prototype application. Central to our approach (see [1]) was the discovery of typical natural language utterances made by legislation drafters (so-called Juridical Language Constructs in short JLC's) that coincide with specific norm expressions. Secondly when formalising these norms knowledge analysts use specific model constructions (so-called translation patterns).

Although the current version of the tool is far from perfect and many extensions to it should be made (and are planned to be made) the current state of affairs shows that the approach is promising. We believe we came a lot closer to realising our original intensions (reducing maintenance costs and reducing time to market).

In this paper we present the approach that we developed by showing how typical legal sentences (parts of actual laws) are translated and represented in a formal model. For a detailed description of the algorithms used and the technical details of our tool we refer to [1].

2 Automated Norm Extraction

All automated support of the modelling process is grouped into the Power-Workbench which is composed of different functionalities that are connected to a central repository. The general outline of the Power-Workbench is given in figure 1. The automated concept and norm extraction tools are integrated into one process denoted as *translate* in figure 1. The concept and norm extraction tool consists of a parser, a lexicon (containing Dutch words and juridical terms), a grammar (containing normal Dutch and specific juridical language patterns), a 'lexicon supplementor' which tries to identify the grammatical category of an unknown word (e.g. a set of digits will be identified as a number, furthermore a combination of nouns that individually are part of the lexicon are considered to be a noun as a whole) and a model generator which translates the parsed source text into formal model components. A modelling interface (the Translate wizard) is added to assist the knowledge engineer to adapt the generated model to suit his needs. The translate tool consists of two separate layers build onto each other. The concept extraction layer is used to generate the domain ontology (or conceptual model) consisting of types, attributes and some associations, expressed in UML. The concept extraction approach is described in [3]. The norm extraction layer is an extension to the concept extraction layer which uses the structures defined for concept extraction.

To be able to automate the norm extraction we need to understand the structure of normative utterances. A previous study (see [4]) leads us to the conclusion that the sentences that occur in legal texts can be grouped in a few categories like definitions, value assignments and conditions. The majority of categories can by described by a limited set of possible juridical natural language constructs (so-called JLC's) used in the sentence. A sentence can be classified by studying the constructs used. A description of this classification is given by de Maat in aforementioned study [4]. Globally normative sentences as they occur in legal texts, i.e. the law, consist of a main sentence and subordinate clauses (which usually add constraints). By using the set of JLC's a substantial part of legal sentence can be recognized and classified by parsing the subsequent language constructs specified in each of the JLC's. After this the translation component can be used to translate the parsed information to create the relevant formal specification language e.g. a UML/OCL model.

To parse sentences in normative texts, the natural language constructs or 'patterns' that can occur have to be described formally. We use a unification grammar to specify the JLC's.

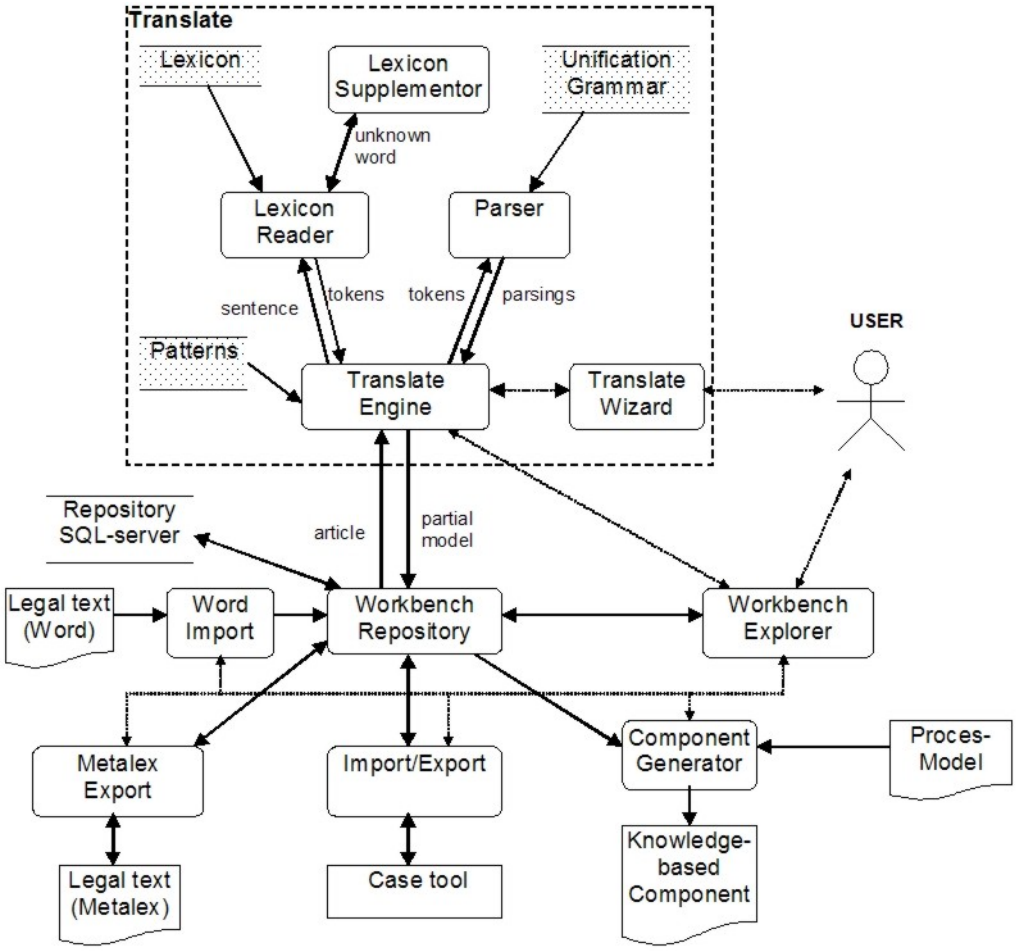

Figure 1: A global model of the Power-Workbench

A comprehensive description of a unification grammar is given by Shieber [6]. Unification grammars provide us with a both elegant and efficient description of the language constructs. Grammar rules are used to describe the general language constructs at a high level of abstraction, while the unification rules are used to enforce agreement between the constituents amongst others, for example the agreement between the subject and the verb of a sentence (e.g. the syntactically correct *you are* versus the incorrect *you is*).

The last important part of the norm extraction tool is the database table with all the so called patterns, which describe the way in which the JLC's are mapped onto the formal modal. In our case these models are built upon the UML/OCL standards [2], [7].

A more detailed description of the Norm Extraction tool is given in [1], [5].

3 Applying the tool to actual legislation

To get an idea of the applicability of the current prototype on actual legislation and regulations, we studied the generated models of some articles from the Dutch Income Tax Law 2001. The articles presented in this paper are not chosen at random, but we deliberately have chosen those articles which contain at least one JLC that would be easy to recognize by the reader. When a sentence does not contain a JLC, the prototype cannot apply norm extraction to this sentence. In that case the prototype's functionality is limited to automated concept extraction, i.e. the generation of types and attributes, where the connection between the types has to be added manually. This is mainly done by looking at noun phrases. A more detailed

description of automated concept extraction can be found in [3].

For each article we will indicate of which JLC's the sentence consists of together with a short explanation on the meaning of these JLC's. We also indicate in which way these JLC's have to be modelled. We will explain both the modelling process and how the NLP-based translation finally will result in a formal model of the given legal text.

The prototype was developed for Dutch texts and consequently the Dutch language and therefore the generated models are in Dutch as well. When we explain the separate articles we will give the original Dutch text as well as an unofficial translation in English. For the actual explanation of the recognition and translation process we will refer to this English translation. To make the discussion understandable we have also translated the original Dutch models in English. These translated models are made by copying the generated Dutch models and translate the texts one to one in English. In the translation no other adaptations are made to the model, so the aspect of automatic generation remains intact.

3.1 Article 2.2. Paragraph 3.

> *Indien een Nederlander op grond van het tweede lid geacht wordt in Nederland te wonen, worden de partner en de kinderen die jonger zijn dan 27 jaar en die in belangrijke mate door hem worden onderhouden, tevens geacht in Nederland te wonen.*
>
> If a Dutchman is deemed to live in the Netherlands based on the second paragraph, the partner and the children younger than 27 years and largely supported by him, are also deemed to live in the Netherlands.

The first example is an article, which contains two different JLC's, namely an *explicit condition* and a *deeming provision*. An explicit condition places a limitation on the main sentence, in this case a deeming provision. A deeming provision is a legal construction in which a statement is considered as true regardless of the real truth value of the statement.

Before the model can be generated, the relevant JLC's in the sentence have to be recognized. For this purpose the prototype contains a parser, which can analyze the sentence by means of a grammar. This grammar gives amongst others a detailed description of the structure of the different JLC's. The description of the JLC's can be done on a high level of abstraction, in which elements defined earlier, like noun phrases, can be used. Because the parsing mechanism is recursive, within a JLC other (or the same) JLC's can be referenced.

An explicit condition appears as subordinate clause in a legal text en is always used to put a limitation on the corresponding main clause. An explicit condition usually starts with the word 'if'. Some alternative words like 'when' are also possible to trigger an explicit condition. An explicit condition has one of the two following structures.

(ec1) If <subject> <condition> , [then]

(ec2) , if <subject> <condition> (.|,)

This structure is given in terms of structures on a lower level of abstraction, in this case <subject> and <condition>. The term <subject> refers to a noun phrase, which is defined in the grammar on a lower level of abstraction. A noun phrase is the most important structure within the concept extraction and is a coherent part of a sentence which consists of a main noun together with its corresponding article, adjectives and prepositional phrases and so on. A noun phrase gives rise to the generation of a corresponding type (and possibly some attributes or other properties) in the formal model [3].

The term <condition> can be replaced by a fixed construction like 'greater than x', but that is not the only possibility. In principle many different language constructs are possible; however each construction needs to state something about the <subject>, where a verb is needed to make the connection between the <condition> and the <subject>. Because the number of possibilities for the <condition> is rather large, we did not specify all language constructs in our grammar, but for the time being we assume that a <condition> consists of a sequence of random words. The drawback is that the prototype has no knowledge of the underlying structure of the phrase, but for the moment we think this does not create a problem. The <condition> will be modelled by creating a Boolean attribute to the corresponding type generated by <subject>. The name of this attribute is created by taking the complete phrase for <condition> hereby removing the spaces and letting each word start with a capital. In our example this will result in the attribute 'isDeemedToLiveInTheNederlands'. Furthermore this attribute must be included as a condition in all constraints which are generated during the handling of the main clause. When we elaborated the structure of the <condition> further it should be possible to generate an attribute like 'livesInTheNederlands', but then there will be no difference between a real and a deemed statement. For the moment we do not know if this distinction is needed or not.

A deeming provision is a sentence in which a given situation is said to be considered as if it where another situation. Thus, if situation A is deemed equal to situation B, then all rules that apply to situation B apply to situation A as well. In this way, definitions can be extended to cover certain special, exceptional situations. A deeming provision can be recognized by the use of the verb 'to deem' and has one of the following two structures.

(dp1) <subject> is <time period> deemed <fiction>

(dp2) is <subject> <time period> deemed <fiction>

Like with the explicit condition the structure of a deeming provision is described in structures on a lower level of abstraction. Note that the second structure (dp2) is a direct translation from Dutch, where this word order is obligatory in combination with (ec1); this word order however is rare or non existent in English, where (dp1) is the preferred structure. As stated before the <subject> refers to a noun phrase and gives rise to the generation of a type in the model.

The term <fiction> describes the situation which fictitiously holds for the <subject>. This is usually done in the form of a statement. For the moment this statement is not elaborated in the grammar, but we consider <fiction> as a sequence of random words. Therefore the prototype has no knowledge of the underlying structure of fiction, but as stated before this should not raise a real problem.

The term <time period> indicates when the <fiction> is valid, for example 'this year' or ' for the application of this article'. We also assume the <time period> to be a sequence of random words.

A deeming provision is modelled by adding a Boolean attribute to the type generated by <subject>. The name of this attribute is created by concatenating the words for 'is <time period> deemed <fiction>', where the spaces are removed and each word starts with a capital, in our example 'areAlsoDeemedToLiveInTheNetherlands'. The corresponding type has to be extended with a constraint in which the value of this attribute is set to true. An alternative could be an attribute with a name like 'livesInTheNederlands', but like before there is no distinction between reality and fiction.

Finally a mechanism is needed to combine both JLC's to form a grammatically valid sentence. This can be done in two manners.

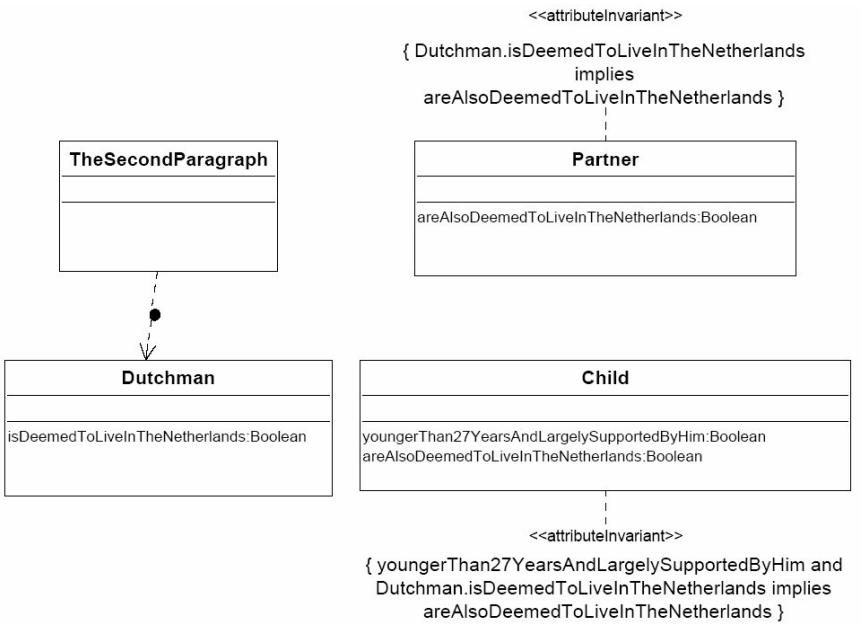

Figure 2: The generated model of Article 2.2. Paragraph 3.

(s1) <explicit condition> <deeming provision>

(s2) <deeming provision> <explicit condition>

Hereby we use the previously defined structures of de JLC's. In addition extra information is added to the grammar to guarantee that a grammatically correct combination is made. In Dutch only the combination of (ec1) with (dp2) is valid for the sentence (s1) and for the sentence (s2) only the combination of (dp1) with (ec2). The generation of the complete model of article is realized by first modelling the separate JLC's and subsequently combine the generated condition from the explicit condition with the generated constraint from the deeming provision. This finally results in the model given in figure 2.

3.2 Article 3.80.

> *Belastbaar loon is loon verminderd met de werknemersaftrek.*
>
> Taxable wages are wages reduced with the employee's discount.

This article contains an example of a value assignment. This type of sentences contains a verb that is used to assign a value to a certain concept. The same verb can also be used to change this value or to compare the value of the concept with another value. The verbs used within a value amount are 'to be' and 'to amount to'. The structure for a value assignment is.

(va) <subject> (is | amounts to) <formula>

The term <subject> refers to a noun phrase and is treated in the same way as in the previous example. The term <formula> in the given structure can take many different forms. It can be a concrete value (e.g. EUR 1000 or 20%), as well as some kind of calculation. Several recursive constructions are possible, such as:

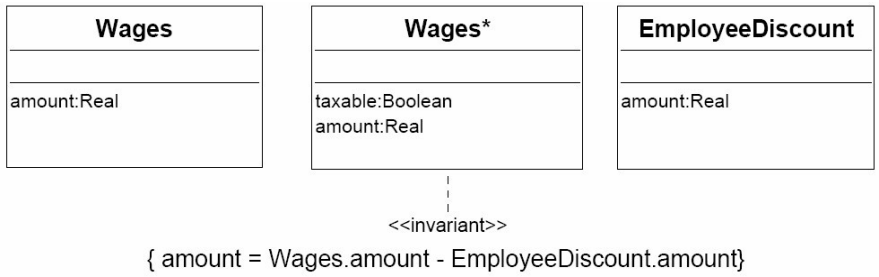

Figure 3: The generated model of Article 3.80.

- x increased by y
- x reduced with y
- at most x
- at least x
- the sum of x and y

These constructions can be translated quite straightforward into an OCL constraint. For example, “x increased by y” would become x + y.

The way in which a value assignment is modelled depends on the noun phrase the <subject> refers to. The standard way to model a value assignment is by using the <subject> to create a corresponding type. An attribute ‘amount’ of type Real is then added to this type as well as an invariant constraint. The text of the constraint starts with ‘amount =’ followed by the translation of the <formula> into OCL. In our prototype noun phrases that depict a value (e.g. ‘the amount of the price’, ‘the height of a tree’) are treated in a special way. These noun phrases generate a Real attribute in the formal model whereas normal noun phrases result in a type. The type this attribute belongs to is usually mentioned in the noun phrase as well (e.g. ‘the height of a tree’ results in an attribute height within the type tree). A constraint is added to this type where the text consists of the name of the attribute followed by ‘=’ followed by the OCL translation of the <formula>. The generated model for the example is given in figure 3.

3.3 Article 1.7. Paragraph 1. Second sentence.

> *Onder een lijfrente wordt mede verstaan de aanspraak op winstuitkeringen voorzover die uitkeringen verband houden met een lijfrente.*
>
> By annuity is also meant the claim on (payments of a) dividend insofar those payments are related to an annuity.

This article is an example of what we call a *type extension*. A type extension is a special legal language construct to broaden or narrow a *definition*. In a definition, a description is given of the concepts that will be used in the legal source. Usually a definition is recognizable by the verbs ‘to be’ or ‘to mean’. A definition is used to define the set of values that the subject of the definition (which will be modelled as a type) can have. For example the sentence ‘a bird is a flying animal’ defines the type bird as a flying animal. We can then use type extension to broaden of narrow this definition. An example of a type extension to broaden this definition is the sentence ‘by bird is also meant a penguin’. The meaning is that the set of penguins is now also valid as value for the type bird. This can be described mathematically by a union; the new set of birds is the union of the previously defined set of birds (i.e. flying animals) with

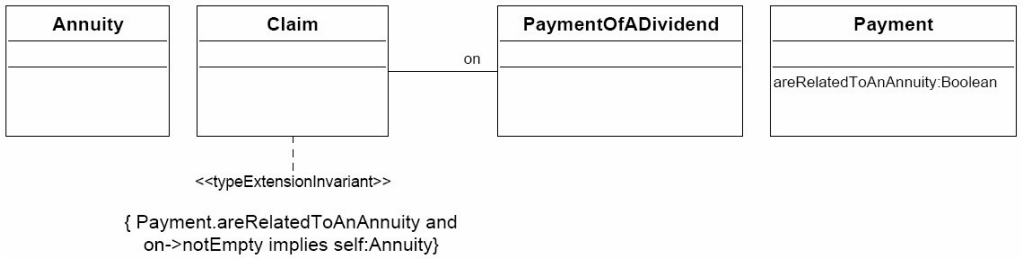

Figure 4: The generated model of Article 1.7. Paragraph 1. Second sentence.

the set of penguins. An example of a type extension to narrow this definition is the sentence 'by bird is not meant a bat'. This can be described mathematically as a set substraction; the new set of birds is the previously defined set of birds (i.e. the union of flying animals and penguins) minus the set of bats.

The chosen article is a case of a broadening type extension. These are characterized by the use of 'to also mean'. The general structure of a broadening type extension is.

(bt3) By <subject> is also meant <definition>.

Both the terms <subject> and <definition> are defined as noun phrases. They are modelled as types by the concept extraction tool. The actual type extension is then modelled by adding a constraint to the type corresponding to <definition> which states that this type is equal to the type corresponding to <subject> (self:<subject>). If necessary, conditions can be added to this constraint (which is the case in our example).

The final model of the article is given in figure 4. A remark shoot be made about this model. In the model we can see the types Payment and PaymentOfADividend which are not related in the generated model, while it is clear from the text of the article that Payment refers to the same type as PaymentOfADividend. This problem arises from the fact that the Dutch text contains two different nouns which are then translated into different types. And although these nouns are used as synonyms in the article they are in fact not synonymous. The only thing we can be fairly sure of is that one noun (PaymentOfADividend) is a specialization of the other (Payment), which could be modelled by a specialization/generalization in the UML model. How and if this is possible must be researched further. Apart from that synonyms should be treated with caution, because words that look like synonyms (e.g. partner and husband) could turn out not to be (e.g. partner meaning associate). Therefore we have chosen to model synonyms as separate types and then manually combine these types into one if possible.

3.4 Article 3.19. Paragraph 5.

Voor de toepassing van dit artikel wordt onder woning mede verstaan een duurzaam aan een plaats gebonden schip.

For the application of this article, a ship permanently tied to a spot is also considered to be a house.

This article is a combination of two JLC's namely a scope definition and a type extension. The meaning and structure type extension is already mentioned in the previous section so we concentrate on the scope definition. A scope definition is used to limit a statement (usually a deeming provision or type extension) to a limited section of the legal source. It has the following format:

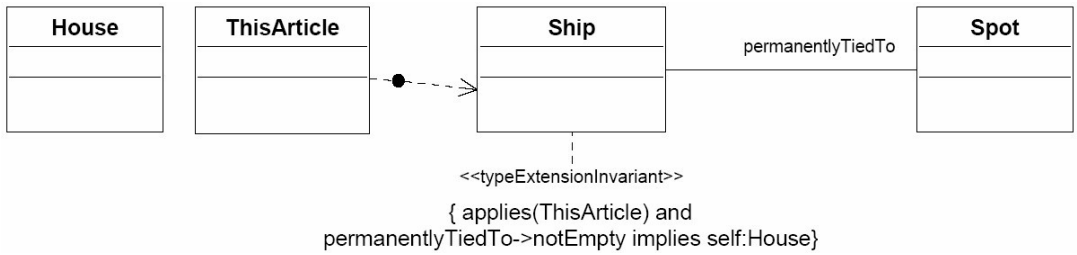

Figure 5: The generated model of Article 3.19. Paragraph 5.

For the application of <reference> <statement>

The term <reference> stands for a reference to a legal source. This could be a relative reference (like 'this article', 'the previous section' and so on) or an absolute reference (like 'article 3.19. Paragraph 5' or 'the Law on Income Tax 2001').

The term <statement> refers to another JLC. In this sentence a type extension: the definition of a 'house' is extended to include a houseboat. This statement has been limited to be only in effect for the application of the reference ("this article") (the part of the legal text referenced by the scope definition).

A scope definition is modelled by creating a PackageReference for the reference. This PackageReference is then connected to the main class of the statement by a dependency relation. Also, the OCL-constraint is extended to include the condition "applies(reference)", so that the statement is limited to situations in which this article applies. The resulting model for our example article is given in figure 5.

4 Conclusions

One of these founding fathers of the field of legal engineering, is the great legal philosopher and mathematician Leibniz, who, in his superior work *Philosophical Essays* stated that "Once the characteristic numbers of most notions are determined, the human race will have a new kind of tool, a tool that will increase the power of the mind much more than optical lenses helped our eyes, a tool that will be as far superior to microscopes or telescopes as reason is to vision.". In the POWER research programme conducted by the DTCA we work in this tradition. We developed and implemented a knowledge based approach and it was showed that explicitly representing norms helps to improve legal quality, reduce time to market, reduce maintenance costs as well as improve legal access for citizens and civil servants. Our desire to reduce the knowledge acquisition bottle-neck and our desire to improve uniformity of the knowledge representation made us look into the fundaments of understanding the semantics of normative expressions in legal texts. Fortunately the syntax of the natural language utterances in the legal knowledge sources (especially law texts) provide us with some handles that allow at least to a certain degree automated extraction of both concepts and norms contained within these normative knowledge sources.

We are very well aware that our NLP-based approach is currently in a prototype stadium and many things still can and must be improved before automated norm extraction can be applied in large scale projects. Nevertheless the results achieved thus far are promising and show that the approach we have developed is a realistic one.

We made plans to extend the current functionality of the extraction tool and test our theory on different legal sources (different Dutch laws and also laws expressed in different natural

languages). We furthermore will look at different formal representation formalisms (which implies that we should change the current translation patterns) like OWL/SWRL.

We also planned to develop NLP-extraction for procedural requirements from legal texts. These requirements should be expressed in a formal process modelling language.

Our long-term ambition is to develop an approach (methods and supporting tools) that helps us to improve the whole chain of processes involved when creating and implementing a normative system (which is the case when new laws are drafted and implemented). We believe our research contributes to this ambitious aims and want to thank all the people that inspired us and especially the European Commission that sponsored the E-Power project (see www.belastingdienst.nl/epower) in the IST 5^{th} framework programme.

References

[1] van Engers, T.M., Sayah, K., van Gog, R., de Maat, E., 2004, Automated Norm Extraction from Legal Texts, Proceedings KDNet Symposium 2004.

[2] Fowler, M., Scott, K., 2000, UML Distilled. A Brief guide to Standard Object Modelling Language, 2nd Ed. Addison Wesley.

[3] van Gog, R., van Engers, T.M., 2001, Modelling Legislation Using Natural Language, Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference.

[4] de Maat, E., 2003, Natural Legal Modelling. Formalising Legal Knowledge using Natural Language Processing, Master Thesis, Universiteit van Twente, Department of Computer Science, Enschede.

[5] Sayah K., Automated Norm Extraction from Legal Texts, Master Thesis, University of Utrecht, Department of Computer Science.

[6] Shieber, S. M., 1986, An Introduction to Unification-based approaches to grammar, Center for the study of Language and Information, Stanford.

[7] Warmer, J., Kleppe, A.,1999, The Object Constraint Language. Precise Modelling with UML, Addison Wesley.