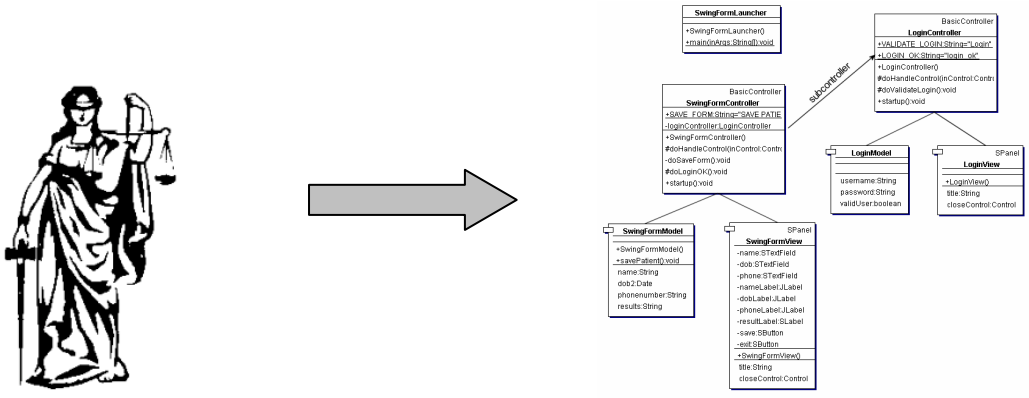
Kamal Sayah

Automated Norm Extraction from Legal Texts



Master's Thesis, Utrecht University,
Department of Computer Science.

Master's Thesis Supervisors:
Prof.dr. T.M. van Engers
Dr. J. Hage

Utrecht, August 2004

Preface

This thesis is the final work of my study at the Institute of Information and Computing Sciences (IICS) at the Utrecht University. During my thesis research, I worked at the Tax Office Utrecht, *Centrum voor Proces- en Productontwikkeling* (B/CPP).

First, I am very grateful to the Tax Office Utrecht, *Centrum voor Proces- en Productontwikkeling* (B/CPP), for giving me the opportunity to write my thesis in a great company in Utrecht. Of course, I would like to use this opportunity to give a special thanks to my main advisor Prof. Dr. Tom van Engers at University of Amsterdam, who has taught me a lot about being a participant of science. I also would like to use this opportunity to give a special thanks to my second main advisor Dr. Jurriaan Hage at the University of Utrecht. He has given me guidance and gave me some critical notes on my work. In that way I kept on a straight line in my research project.

Special thanks go out to Ron van Gog, employee at the Tax Office Utrecht (B/CPP) who was always willing to help and assist me during my thesis research and implementation.

I also want to thank some other people who have supported me in times of need. First, great thanks go out to my girlfriend, parents and sister. Furthermore, I want to thank my fellow employees and thesis students at the Tax Office Utrecht, Rob Dijers, Philipe de Lang, Arian Jacobs, Vincent van Dijk, Niels Egberts, Wilco Niessen and Faridah Liduan for making it possible to have a great time during my thesis research.

Parts of this master thesis have been published in an article in the proceedings of the KDNet 2004 Conference (KDNet Symposium: "Knowledge-Based Services for the Public Sector", [32]). This article (Van Engers, T.M., Sayah, K., Van Gog, R., De Maat, E., [33]) will also be included in a book on that conference, yet to appear.

Abstract

Within the (E-)POWER research program at the Tax Office Utrecht 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 (E-)POWER program, automated concept extraction techniques and a type model generation tool have been developed to improve modelling productivity. Formalization of the norms has thus far been a manual process. In this thesis, a description is given of the development of a further step in overcoming the knowledge acquisition bottleneck: a complete algorithm for automated norm analysis from legal texts. This algorithm makes use of invariant linguistic structures at the syntactical level that characterises specific normative expressions in natural language.

Content

1 Introduction

Governments and many other organisations have often to deal with many regulations and business rules. These are often expressed in natural language and sometimes their volume and complexity are a burden for these organisations. Many processes are affected when new or adapted regulations have to be implemented and both the organisations responsible for the implementation of the regulations and their clients will benefit from a design methodology well suited for this. Within the (E-)POWER ((European) Program for an Ontology based Working Environment for Regulations and Legislation) research program (a project of the Tax Office Utrecht) a first version of such methodology has been developed (see e.g. Van Engers & Boekenoogen 2003 [1]).

The (E-)POWER approach is concentrated around the processes that are involved in implementing normative knowledge sources, i.e. legislation, regulation or business rules etc. This will result in operational processes, which subsequently are based on these regulations, described in the aforementioned normative knowledge sources and on the policy, which may be influenced by business economical considerations. These operational processes form the implementation of a normative system (a system that states what is prohibited, should be done or is allowed). The normative knowledge sources themselves are created in the political arena and the process of creating such knowledge is embedded in a social environment (see Figure 1).

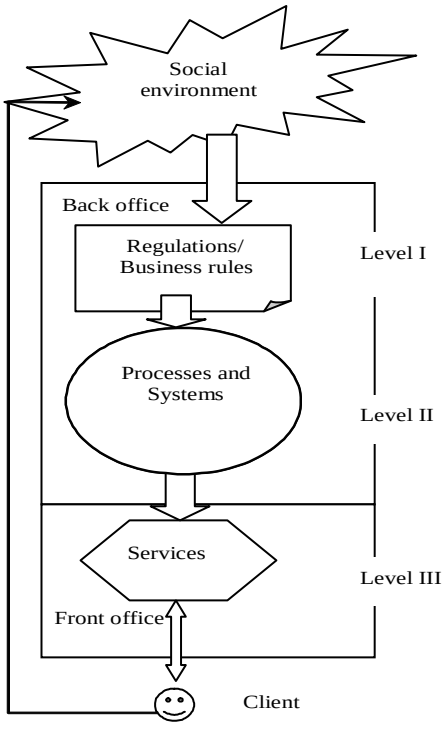


Figure 1. *General interoperability framework for normative systems (from the presentation at KDNet Symposium [32]).*

The design methodology developed in the (E-)POWER project follows the steps from the normative knowledge sources represented in document form, via the

generation of a relevant formal model to a knowledge-based component (i.e. a piece of software able to make inferences about a certain regulatory domain). Figure 2 shows us the different steps followed by the (E-)POWER project.

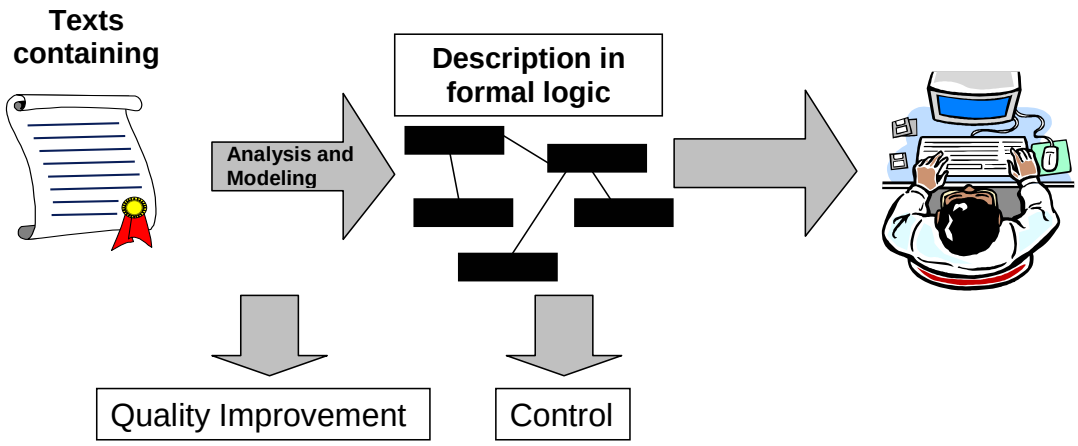


Figure 2. *Aims of the (E-)POWER approach (from the presentation at KDNet Symposium [32]).*

Each of the aforementioned steps consists of a specific approach aimed at solving problems that come with the transformation from one form of knowledge into another (e.g. translating a sentence in a piece of law text into a formal expression or model containing the norm addressed in that piece of text).

As may be expected from any design methodology, these steps should be repeatable and transparent. The formal descriptions created using the (E-)POWER approach can be used as a basis for creating the operational processes and supporting (knowledge-based) systems.

The first step (the generation of the formal models from normative sentences) is also known as "automated norm extraction from legal texts". During this thesis research, one of the subproblems of this first step, verb-phrase extraction, will be tackled.

This thesis is based on the preliminary research done by De Maat 2003 [6]. De Maat has tried to formalise legal knowledge using natural language processing by introducing a (limited) set of predefined natural language constructs (in this thesis they are called JLC's (Juridical (Natural) Language Constructs)), which can be used to define a subset of all possible legal sentences. During this thesis research, this knowledge is used for the generation of the formal models. Because the set of JLC's is not yet complete, this approach will also be limited. After this thesis project, the possibility arises to extend the legal knowledge (set of JLC's) by further examination of the complete set of legal sentences. In the end the ePower Workbench, the relevant project started by (E-)POWER, will have all the knowledge to translate the complete set of legal sentences into their relevant formal model.

When using the legal knowledge (the different JLC's) a legal sentence can be classified to one or more JLC types. This is done by parsing according to a set of production rules, which consist of normative elements. Considering each of the production rules we determine which of these matches with the current sentence. After this classification step, the relevant normative information is extracted from the

legal sentence. When a single classification is not possible, feedback has to be provided. In case of multiple classifications (i.e. in the presence of ambiguity), the user will be asked to make choice and in case no classification is possible then either the legal sentence is incorrect or it cannot be classified according to the existing set of production rules. As a result of the latter, some production rules have to be changed or new ones have to be constructed. The user can make these relevant changes, so afterwards this legal sentence can be recognized (see Section 16.11). Ambiguities can be handled in various ways: the user can decide in an ad hoc way, which classification is to be preferred, or he can decide to refine the existing production rules to distinguish between the classifications.

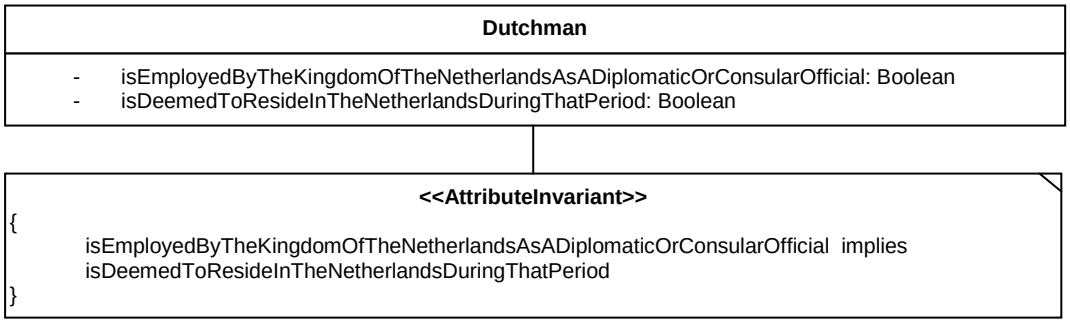
The next step is the translation of the normative information into a formal model (in this case expressed in UML/OCL). Special translation patterns are necessary. These patterns make use of the parsed information and translate this into the formal model. The next example illustrates this idea.

IB 2001 art 2.1 member 2.2
A Dutchman who is employed by the kingdom of the Netherlands as a diplomatic or consular official **is deemed** *to reside in the Netherlands during that period.*

Application of the JLC Deeming Provision

<subject>**[is]**<denotation of time period>**[deemed]**<fiction>

Application of the Translation Pattern



The above example shows us a legal sentence, which can be classified to the JLC type Deeming Provision, by application of the relevant production rule. The constituents in the example production rule are called normative constructs (i.e. <subject>, [deemed]). The normative elements are those parts of the legal sentence which match with the normative constructs (i.e. “A Dutchman who is employed by the kingdom of the Netherlands as a diplomatic or consular official”, “deemed”). The next step is the generation of the formal model. This is done by application of the translation pattern for the JLC type Deeming Provision. In the example the isDeemedTo-part of the attribute isDeemedToResideInTheNetherlandsDuringThatPeriod: Boolean seems unnecessary because of the existing implication (see the AttributeInvariant). Nevertheless, in our implementation the verbs are also concatenated with the other

attribute parts. The normative information within each of the attributes are not taking into account[1] so at this moment this seems the best possible solution.

Chapter 2 gives a description of the internal organization of the Tax Office Utrecht and the ePOWER project. Chapter 3 is used to describe the main problem and the different hypotheses. In addition, the general purpose of automated norm extraction from legal texts is described in this chapter. In Chapter 4, a detailed description is given of automated norm extraction. Chapter 5, 6, 7, 8, and 9 describe which techniques are used in this thesis project in order to implement a tool for the second step, verb-phrase extraction, for automated norm extraction from legal texts. Chapter 10 describes the treatment of some special normative constructs (values, inheritance and the generation of class-names, fixed noun-phrases). In Chapter 11, an alternative solution for the main problem is described. Chapter 12 and 13, give a description of the implemented components added to the ePOWER Workbench for efficiency reasons.

Also, in these chapters some optimisations (alternative solutions) and adaptations to the preliminary research, done by other members of the (E-)POWER project, are discussed. At the end of this thesis some recommendations for future research are made. In addition, some related work (alternative specifications for this specific problem) is mentioned.

[1] Maybe in the nearby future some better knowledge becomes available of the normative information within the attributes of the different JLC's, so a more practical formal model can be constructed.

2 The Dutch Tax and Customs Administration

My thesis research has been done at the Tax Office Utrecht, Centrum voor Proces- en Productontwikkeling (B/CPP)[2] as the final project of my study at the Institute of Information and Computing Sciences (ICS) at the Utrecht University. In the next couple of paragraphs a short description will be given about the different parts from which the Dutch Tax and Customs Administration is composed of.

The Dutch Tax and Customs Administration is part of the Ministry of Finance, which is subdivided in a couple of individual organisations:

- Directorate -General Dutch Tax and Customs Administration (DGBel)[3]
- Tax- and Customs Offices
- Facility Centres
- A Detection facility

The first organisation is the staff of the Directorate -General Dutch Tax and Customs Administration. Together with the Internal Accountancy Directorate (IAB)[4], they are responsible for the financial accountability of the Dutch Tax and Customs Administration. They also give advice about the different aspects of the internal management.

All offices, so the executive facilities (like the Tax- and Customs offices), and the DGBel (Directorate -General Dutch Tax and Customs Administration) are supported by the different Facility Centres (technically and informally). The Facility Centres take care of all the services and applications used to support the employees in fulfilling their daily tasks. One can think of internal/external communication services, network control and the development of services for supporting and improving the internal business processes. One of the Facility Centres is the so called B/CPP (Centrum voor Proces- and Productontwikkeling). This facility centre develops new processes and products, which are used by the Dutch Tax and Customs Administration to pursue their targets and strategies (initially for supporting the primary process). One can think of the translation of the legislation to computational models (which is the development where I am taking part of during my thesis research), designing and modelling of the logistical process or the improvement of existing products and processes.

All employees working at the B/CPP work, dependent on their knowledge area (normally they are called experts in one field of science), in one of the twelve so called domains. Every domain is managed by a domain manager. Every domain can

[2] The English name for the "Centrum voor Proces- en Productontwikkeling" is "The Centre for Process and Product Development".

[3] The Dutch name for the "Directorate -General Dutch Tax and Customs Administration" is "Directoraat-Generaal Belastingdienst" also known as the DGBel.

[4] The Dutch name for the "Internal Accountancy Directorate" is "Interne accountantsdienst belastingen" also known as the IAB.

be seen as some kind of Job Centre[5] from where the employees are dispatched to projects within their area of expertise, and beside this also fulfilling their daily tasks. One of the domains, from where this thesis research has been started is the Vakontwikkeling Klantbehandeling[6] domain. One of the projects started by this domain is the POWER project (Program for an Ontology based Working Environment for Regulations and Legislation)[7].

A couple of years ago the POWER project started a project called The ePOWER Workbench (see Workbench 2.6 2000 [2], and USER DOCUMENTATION ePOWER Workbench 2.6 [3]. The main target of this project (and of the (E-)POWER project) is developing a tool to support one of the steps (see Chapter 1) for the generation of knowledge components (from normative knowledge sources represented in document form, via a formal model to a knowledge-based component (i.e. a piece of software able to make inferences about a certain regulatory domain)). The final tool can be seen as some starting point for the implementation of normative reasoning applications (applications that have the ability to reason about cases). In the next sections I will discuss the detailed aims and objectives of the (E-)POWER project (see Organisatie van de Belastingdienst 2004 [4] for a more detailed specification of the internal structure of the Dutch Tax and Customs Administration).

2.1 ePOWER: European Program for an Ontology based Working Environment for Regulations and Legislation

Within this section a description is given about the (E-)POWER project. The information is collected from the (E-)POWER web page (see E-POWER Homepage [5]). For more information about the (E-)POWER project visit this homepage.

2.1.1 Project Description

The E-POWER project is supposed to achieve at least the following results:

- Developing a method, and supporting tools, with which legislation can be 'translated' into formal specifications that can be used by computers.
- Developing a pension server for the (European) citizens with which they will be able to analyse their own pension regulations.

Since the project will apply the same method to both Belgian and Dutch (pension) legislation and regulations it will be possible to compare these two types of legislation and analyse the differences.

One of the objectives is furthermore not only to make this domain more transparent for the citizens but also for example for insurance companies that offer pension arrangements. The Netherlands aim to open up their pension market to foreign companies but these companies will have to meet requirements. The analysis with the E-POWER method also strives to give insights into these requirements. In

[5] The Dutch word for "Job Center" is "Uitzendbureau".
[6] The English word for "Vakontwikkeling Klantbehandeling" is "Development of Client Handling Processes".
[7] The Dutch abbreviation "POWER" stands for "Programma Ondersteuning Wet- en Regelgeving".

this way it should provide an instrument, which could decrease cross border obstacles for pension providers.

The opening of the Dutch market will also expand the options for Dutch citizens. They will be able to choose providers from different countries.

Concluding: E-POWER aims to realise a method[8], but also products (e.g. E-services such as the pension server). The applicability is not confined to just one of pension legislation and regulations, but directed at all three.

2.1.2 Project Objectives

E-POWER will implement a knowledge management solution by providing a method that help to improve the quality of legislation while the enforcement of law is being facilitated.

This method will decrease the time to market new/changed legislation, facilitate the maintenance of legislation, and it will improve the access to the governmental body of knowledge by offering new E-services.

Furthermore, the use of this method will result in a more efficient use of scarce knowledge resources. The E-POWER project will result in transparency of pension arrangements for the (future) elderly citizens.

The project will offer tools that help with the harmonisation of pension regulations. By providing easy access (using the Internet) to vital information the project will contribute to the social inclusion of citizens. E-POWER will consequently improve the effectiveness and efficiency of public administrations and contribute to the completeness of the internal market.

[8] When in the rest of this chapter the word "method" is used also "method and tools" can be used.

3 Problem Description

In the introduction (Chapter 1) it is mentioned that the global approach of the (E-)POWER project consists of a couple of subsequent implementation steps (from normative knowledge sources represented in document form, via a formal model to a knowledge-based component, see Figure 2). The first step towards the generation of knowledge components is the generation of formal models from normative texts (legal sentences). For the generation of these formal models a special tool has to be developed. In Chapter 2, the relevant tool, which is part of a production environment, is called The ePOWER Workbench[9]. The main problem of generating formal models from legal sentences can be subdivided into two sub problems. The first one is the recognition and translation of noun-phrases (concepts) from legal texts (noun-phrase extraction), and the second one is the recognition and translation of verb-phrases (verb-phrase extraction). When I started my thesis research the norm extraction tool (part of the ePOWER Workbench) had all the functionality needed for the noun-phrase extraction step.

My task was to extend the ePOWER Workbench norm extraction tool with all the functionality necessary for the second step, verb-phrase extraction, of the generation of formal models. Finally, the norm extraction tool will have all the necessary functionality needed for the generation of knowledge components. Of course, afterwards we have to check that we have preserved the intended meaning of the initial legal sentences. This is done by inspecting (by hand and brain) the outcome of the translation process.

A starting point for this thesis research is the availability of a subset of all legal sentences described in the Dutch legislation in a form that they can be used by a computer to reason about. This means that these legal texts are built upon some kind of template (Word-documents[10] subdivided in structure blocks), so they can be used as input for the final norm extraction tool (the translation engine of the ePOWER Workbench).

During my thesis research, I made use of the results of some research done by De Maat 2003 [6]. De Maat has tried to formalise legal knowledge using natural language processing by introducing a (limited) set of predefined natural language constructs (in my thesis research known as JLC's (Juridical Natural Language Constructs)), which should define all possible legal sentences (i.e. legal norms). These JLC's could consequently be used to extend the norm extraction tool with functionality necessary for verb-phrase-extraction. In Chapter 5, the usability of those JLC's will become clear.

Because, this thesis research is a continuation of the initial research done by De Maat the extension of the ePOWER Workbench translation engine is limited to the recognition of a subset of the complete Dutch legislation. Within this research only a

[9] Another name for the ePOWER Workbench tool is the *Norm Extraction Tool*. Both names will appear and be used to explain something in this thesis, so both have the same meaning.
[10] Word-documents (Microsoft Office Word 2003 [7]).

single law has been partly translated into JLC's. This is the law on income taxes from 2001 (Wet Inkomsten Belasting 2001, IB2001 [8]). Therefore, the examples used within my thesis are similar to the examples used by De Maat. Of course, during the test phase of my implementation some other legal sentences are used.

3.1 Hypothesis

In my thesis research the main problem is to find a way to recognize verb-phrases and extract their normative content and then translate them into their relevant formal models (in this case expressed in UML/OCL, see Fowler, M., Scott, K. 2000 [9] & Warmer, J., Kleppe, A. 1999 [10]). Initially, the concept extraction tool has functionality to extract and translate noun-phrases as mentioned in previous sections. My task was to extend the initial ePOWER Workbench with the functionality needed for the second step towards automated norm extraction (verb-phrase extraction).

My thesis research is a continuation of the thesis research done by De Maat 2003 [6]. Therefore my approach is built upon the legal knowledge (the knowledge about how legal sentences can be formalized by using natural language processing) described in a form of a (limited) set of predefined natural language constructs (JLC's). By examining each of these JLC's (the global structure of the JLC's) I can generate special parse rules for extracting the relevant information from the legal sentence (the recognition step) and afterwards generating a translation pattern for the generation of the relevant formal model (the translation step).

Thus, for the recognition step I will test the following hypothesis:

> *When examining the (limited) set of predefined natural language constructs (JLC's) defined by Emiel de Maat, special parse rules can be generated to extract the necessary legal knowledge from the legal sentences.*

And for the translation step I will test the following hypothesis:

> *After the application of the parse rules, special translation patterns can be applied to generate the relevant formal models (expressed in UML/OCL).*

3.2 The General Purpose

The formal models used in the (E-)POWER approach are expressed in UML/OCL (Unified Modelling Language/Object Constraint Language, see Fowler, M., Scott, K. 2000 [9] & Warmer, J., Kleppe, A. 1999 [10]). Van Engers and Glassée 2001 [11] describe the way legal source texts are translated into UML/OCL models. These UML/OCL representations of the legal texts have shown to be quite suitable in different projects. Creating these formal models however is a time consuming activity, and has to be done by high-skilled staff (according to the (E-)POWER project members experience it takes an experienced knowledge analyst approximately 1.5 days per A4 law text). As a regular approach to be used in large organisations the

productivity of the modelling process has to be improved since the global aims of the (E-)POWER approach are three (see Section 2.1):

1. Reduction of implementation time (time to market)
2. Improvement of the quality of the normative knowledge (or legal quality)
3. Reduction of total cost of ownership (of the normative systems that result from the implementation processes).

To improve the productivity of knowledge analysts in their job, in the (E-)POWER program a research project on using natural language processing (NLP) was started to analyse normative texts expressed in natural language and for the generation of (parts of) the formal models that contain the normative knowledge in a form that is suited for building knowledge-based components (programs that have the ability to reason about cases). So, the logic that implicitly lies beneath the regulations becomes explicit.

The fact that legislation and other normative texts are written in natural language causes some difficulties. The most important problem is that natural language is ambiguous, which entails that an expression in natural language can have multiple meanings. This problem can arise both at word level, for example 'bank', and at phrase or sentence level, for example 'it is not allowed to shoot a man with a gun'[11]. Another problem that arises is that natural language contains vague and unclear notions like 'almost', 'for the most part' and typical juridical open evaluative terms like 'justified' or 'good practice'. When using legal texts (acts, norms etcetera) there are also some anomalies.

When we look at the following characteristics of legal texts this statement becomes clear:

- Legal texts are the explicit results of a group-dynamic process.
- They contain norms that express what is obligated, permitted and allowed.
- These norms reflect underlying preferences and value systems
- Legal texts can be perceived as specifications for normative systems.
- Legal texts are under specified.
- They suffer from anomalies: inconsistencies, circularities, open evaluative terms and vagueness.

So, before normative reasoning (e.g. in law enforcement) can be automated, legislation has to be translated into a language that does not have these aforementioned problems and can be read by a computer, i.e. a formal specification language, in our case expressed in UML/OCL. Governments can furthermore benefit from the fact that ambiguous constructs in the law can be detected during the translation into such a specification language. This is especially the case when the

[11] This sentence has two possible interpretations. In the first place, one can interpret that it is not allowed to use a gun to shoot a man. The other can interpret that it is not allowed to shoot a man who has a gun. The difference in interpretation lies in the attachment of the preposition (also known as the PP-Attachment problem).

translation is done at an early stage in the chain of processes needed to implement new or adapted regulations. Than it is often still possible to change ambiguous constructs, vague terms etcetera, before the law becomes enacted (since repairing unintended meaning in a later stage is much more expensive). When changes are not possible or desirable, one could still provide the law enforcement organizations with non-ambiguous interpretations (which of course should be documented as additional knowledge sources). In the ideal situation, normative texts should be non-ambiguous.

4 Automated Norm Extraction

In this chapter, the complete process of automated norm extraction is described. Figure 3 visualizes this complete process. It shows us the general approach towards automated norm extraction. The arrows in the figure are used to indicate the subsequent steps for the generation of the knowledge components. The implemented tool accepts Legal Sources[12] as input for the translation step. The translation step results in two possible models: Process models and Conceptual models. The Process models can be transformed into Task models (for a detailed description of how to use UML/OCL for expressing process and task knowledge, see Egberts 2004 [12]). From both the Conceptual models and Task models, the final Knowledge components can be generated. The dashed arrows are cyclic arrows (the model or the component leads to a new legal source, which can be translated again).

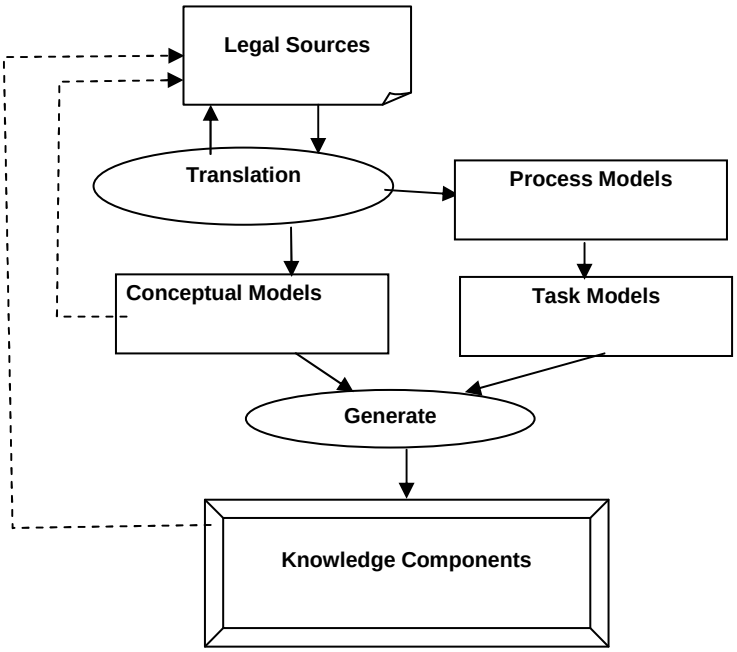


Figure 3. *The general approach towards Automated Norm Extraction (from the presentation at KDNet Symposium [32]).*

For the complete automated norm extraction two different implementation steps are used, namely noun-phrase extraction (the extraction of concepts) and verb-phrase extraction (the extraction of meaningful coherent structures in legal sentence). Both describe functionality, which can be used within the translation step to generate the relevant (formal) models. Later on, we can use these specific models to generate knowledge components, which can then be used in knowledge applications.

4.1 Noun-phrase Extraction

The first implementation step has resulted in an automated concept extractor, which allows a computer to identify the different concepts (noun-phrases) that exist

[12] The input texts are written in Microsoft Word based on a predefined template.

in a given legal sentence (see Van Gog & Van Engers 2001 [13]). This step is largely based among others upon the Object Modelling Technique (OMT) (see Rumbaugh 1999 [14], Frederiks 1997 [15]). From a computational linguistic point of view this is a simple method to implement. The general disadvantage of this method, however, is that generally too many object types are introduced. This is not a problem when using an automated concept extraction tool. Rumbaugh, not benefiting from the availability of an automated extraction device, gives a few guidelines for reducing the number of object elements (types, attributes, operations and roles/relations). These guidelines aim at reducing redundant, irrelevant and vague object elements as well as implementation constructs. Generally speaking, world knowledge is needed to determine whether these criteria apply. Considering world knowledge however is not desirable, because the possibility exists that extra information is introduced into the model that we want to derive from the original legal source texts and of course we don't want this model to contain any knowledge that is not part of that normative knowledge source. If the use of world knowledge can't be avoided, that knowledge should at least be explicated (and documented as a separate knowledge source).

One can easily understand the benefits of being able to extract the concepts that are used in a specific piece of law text. For various reasons it is important to have insight in the concepts that are used for legal reasoning. Legislation drafters could use the insight in existing concepts to decide on the reuse of those concepts. Sometimes introduction of new concepts is needed or specialization is needed for being able to express the normative statements (e.g. expressed as an article in the law), but if an already existing concept can be reused this will reduce the implementation effort. Furthermore, reduction of administrative costs for citizens and business are one of the interests for the legislator. Insight in the concepts used in a piece of law can therefore be used to calculate administrative costs.

From a knowledge engineering perspective, the automated concept extractor has further benefits. Not only it reduces the amount of work that knowledge analysts normally have to do, but since automated generation of models also increases inter-analyst independency its application results in more uniform models as well. More uniform models are easier to understand, easier to process when creating applications based on these models and easier to maintain.

4.2 Verb-phrase Extraction

The second step towards automated norm extraction consists of modelling elementary sentences (later named as JLC's) in the original normative knowledge source, optionally using the results of the first step. This step is partly comparable with the NIAM approach by Nijssen 1989 [16]. The disadvantage of NIAM is that the text must be transformed into elementary sentences, which is quite labour intensive. The modelling of elementary sentences can therefore just be used as an intermediate step and it must always be possible to trace the final model back to the source text.

In my approach, I try to identify and model meaningful coherent structures in the text in a pragmatic way. The tool, which has to be developed, should not be limited to the modelling of nouns, but on the other hand it does not have to model complete

(elementary) sentences. Therefore, I concentrate on syntactic constructs (invariant linguistic structures at the syntactical level that characterize specific normative expressions in natural language) and translation patterns (describing how natural language constructs should be translated into constructs in the final formal UML/OCL model). I use standardized transformations to translate the relevant legal constructs found in the legal sentence into a formal expression or model, thus enhancing uniformity i.e. inter coder independency. The approach is therefore a 'middle-out' approach.

5 The Categorization of Legal Sentences

A previous study (see De Maat 2003 [6]) leads us to the conclusion that the sentences that occur in legal texts can be grouped in a few categories (see Figure 4):

- Nouns
- Definitions and Type extensions
- Scope Definitions
- Deeming provisions
- Application provisions
- Value assignments and changes
- Conditions
- Norms

For each category (with the exception of norms which are expressed in one or sometimes more than one natural language sentences), there is a limited set of possible juridical natural language constructs (so-called JLC's) used in the sentence. I refer to Appendix A, for the complete set of global structures belonging to each of the predefined JLC's.

A sentence can be classified by studying the constructs used. A description of this classification is described by De Maat 2003 [6][13].

Globally normative sentences as they occur in legal texts, i.e. the law, consist of a main sentence (seven possible types indicated in Figure 4 in italics) and subordinate clauses (which add constraints). In addition, more extensions exist.
The fat boxes in Figure 4 denote elements that are always present as part of the higher-level element, while the thin boxes are optional extensions. The simplest form of a sentence consists of just a main sentence, which includes one or more noun phrases that are only composed of their main term (the noun), its article and any adjectives that are not considered part of an implicit condition. Although the treatment of adjectives has not been described by De Maat we state that adjectives can be treated as implicit conditions but also as part of a term.

An example is the noun phrase "taxable income", which can be interpreted as a single term, or as a term with a condition (the term "income" with a Boolean attribute "taxable" and a condition "taxable = true" (in our tool the human expert can decide what translation suites him best, but both interpretations are correct from a modelling point of view). Sentences that are more complex are created by adding other elements. For example, a more complex sentence can be created by adding a condition (optional element, thin box). This condition will at least consist of an actual condition (necessary element of condition, fat box).

[13] The global structure of normative sentences (see Figure 4) is a subset of all possible existing structures within a legal sentence. Further research is necessary to specify new structures (JLC's) but at this moment this can be seen as one further step towards formalizing legal sentences.

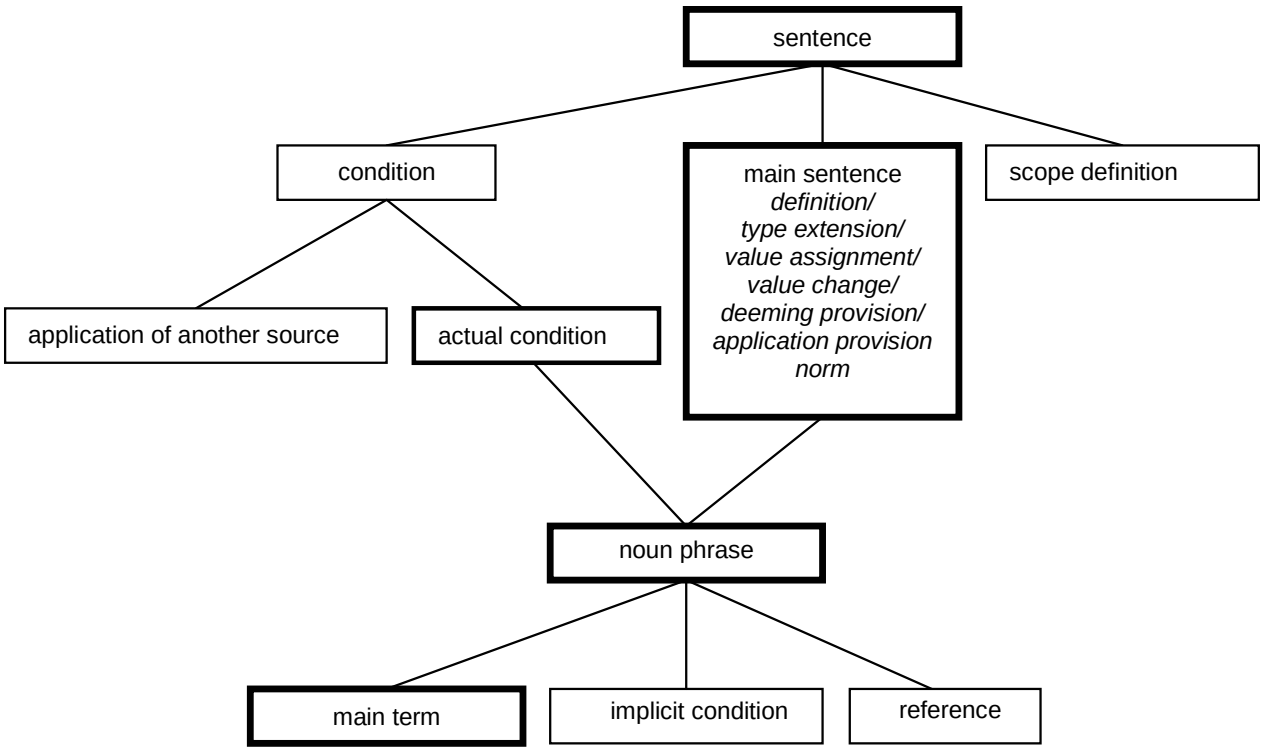


Figure 4. *The global structure of normative sentences as they occur in legal source texts (master's thesis De Maat 2003 [6]).The user can compose a legal sentence by looking at the global structure. An element described in a fat box is always present as part of a higher-level element and the thin boxes are optional extensions.*

By using the set of JLC's any legal sentence[14] can be recognized and classified by parsing the subsequent language constructs specified in each of the JLC's (see Figure 5).

IB 2001 art 2.1 member 2.2
A Dutchman who is employed by the kingdom of the Netherlands as a diplomatic or consular official is deemed to reside in the Netherlands during that period.

⇩

`<subject>` **`[is]`** `<denotation of time period>` **`[deemed]`** `<fiction>`

Figure 5. *Example of the subsequent structures specified in the Deeming Provision JLC*

After this, a translation component, which consists of a set of translation patterns, can be used to translate the parsed information to create the relevant formal specification language e.g. expressed in UML/OCL.

5.1 Parsing Normative Texts

Since all categories of legal sentences (see the beginning of this chapter) except norms can be identified by their specific language constructs, a computer can test for their presence. Consequently, automated classification (parsing) is possible.

[14] Of course, I mean the legal sentences chosen and used within the preliminary research done by De Maat 2003 [6] (law on income taxes from 2001, IB2001 [8]).

Initially, I thought of making use of the possibility to specify the juridical information for each of the different words in a lexicon table. The information is stored in a database (easily extendable, administrable). By specifying an algorithm (for example pattern matching) which examines the legal sentence for the occurrence of these different juridical structures, we can classify the legal sentence into a well known format. However, for the generation of the formal models we need other language constructs to specify the formal attributes, relations and associations. In addition, when we look at the structure of the JLC Definition 1 (`<subject> <is> <definition>`) the main term *is* of the structure is not sufficient to classify to one specific JLC. It also can generalize to other types. See Section 11.1 for a more detailed description about this dynamic approach.

So, in order to be able to parse sentences in normative texts, the natural language constructs or 'patterns' that can occur have to be described formally. I use a unification grammar to specify the JLC's. Shieber 1986 [17] gives a comprehensive description of a unification grammar. 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 syntactic level, while the unification rules are used to enforce agreement between the constituents.

An example is that given the grammar rule `S -> PRONOUN VERB`, both the sentences "you are" and "you is" are correct according to this rule, while it is clearly true that the latter is not correct. By adding the unification constraints `PRONOUN.person = VERB.person` and `PRONOUN.number = VERB.number` to the rule the latter is excluded. In this example, six rules would be needed without unification, namely one rule for each combination of person and number.

Generally parsing of natural language is problematic because often natural languages are highly ambiguous and the meaning of concepts can only be understood looking at their context (see Van Engers & Glassée 2001 [11]). A legal source should be syntactically unambiguous, so ambiguity should not be a major problem in our case. Although we cannot guaranty that the legal text is hundred percent unambiguous, we can assume that syntactical ambiguity is unintended. Therefore, the parsing process is supervised, so when a syntactical ambiguity is encountered, all the different alternatives (i.e. the parse trees) will be presented to the user who subsequently can select the most appropriate one. When such ambiguous translations are possible, feedback to the legislation drafter or legal experts is desired, because ambiguity is something we definitely want to avoid.

Finally, it is not necessary to recognize the entire sentence; it is sufficient to parse the top levels. This is because (in the (E-)POWER project) parts of the sentences are kept in one piece. For example, a deeming statement has the following pattern: "noun-phrase [wordt] time-period [geacht] fiction". When building a model of this text, the entire fiction and time-period language constructs (the largest part of the sentence) are often kept in one piece (in almost every case these pieces are translated to OCL attributes by simply concatenating each word). For the generation of class-attributes see Chapter 9.

6 The Parts of the Norm Extraction tool

The norm extraction tool (as it is present in the beginning of my thesis research) consists of a parser, a lexicon (containing Dutch words and their optional 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 (also known as the set of translation patterns) which translates the parsed source text into formal model components. A modelling interface (the Translate wizard, see Figure 6) is added to assist the knowledge engineer to adapt the generated model to suit his needs.

The norm extraction tool is used to generate the domain ontology (or conceptual model) consisting of types, attributes and relationships, expressed in UML.

Van Gog & Van Engers 2001 [13] describe the concept extraction approach. Since in the end the ambition of this research is to support the formalization of the normative content of legal text, we have to go a step further than just concept extraction.

First, let us take a closer look to each of the different language dependent parts of the ePOWER Workbench, which are stored in the translate-nl[15] database.

[15] Unfortunately, the existing norm extraction tool (the Workbench) only consists of the Dutch language dependent parts. Because these parts are stored in separate database tables one can imagine that adding more language support is trivial.

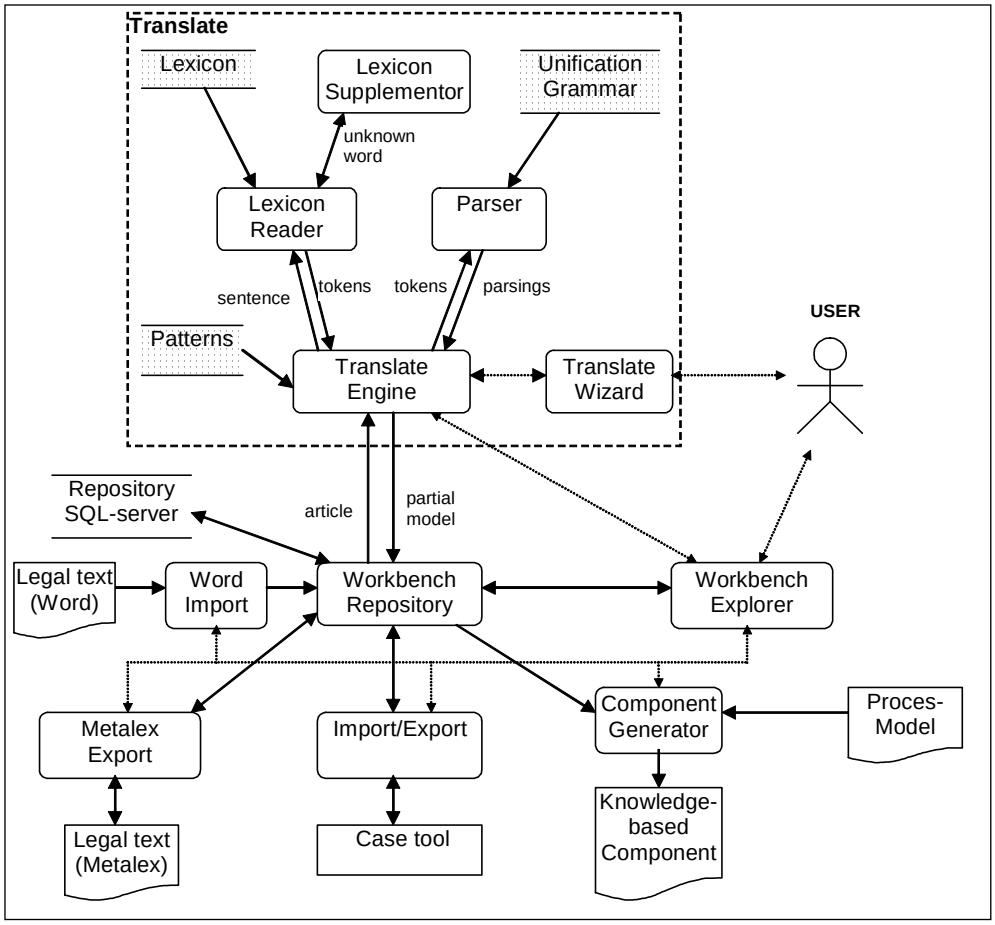


Figure 6. *The global model of the ePOWER Workbench. The ePOWER Workbench is composed of different functionalities that are connected to a central repository (Van Engers, T.M., Sayah, K., Van Gog, R., De Maat, E. 2004 [33]).The relevant elements for this thesis project are dotted. The other parts are just to visualize the complete Workbench Project, so no further description is given.*

6.1 Lexicon

The lexicon is the most standard part of the ePOWER Workbench, because this part is filled with information, which is derived from different language institutes. Globally, the lexicon is a description of all possible words and word shapes with their lexical meaning.

Id	Sem	cat	head. subcat	Root	head. agr. gen	head. agr. case	head. mood	head. Tense	head. agr. Per	head. agr. num
6	**deemed**	V	MAIN	**to deem**			INDICATIVE	PAST	2	P

Table 1. *A record from the lexicon database table*.

Table 1 presents an example of a record corresponding to the word 'deemed' as it is stored in the lexicon table. The table consists of a couple of columns, each representing some grammatical meaning, also called features.

Figure 7 presents a global tree structure of the features, which are stored in the lexicon table. These features can be used during the parsing process for the selection and extraction of the correct grammatical language constructs from the legal text. Later on, within the description of the Unification Grammar (see Section 6.2) it will become clear how this stored feature-set can be extended by describing new features (Grammar rules) within each production rule.

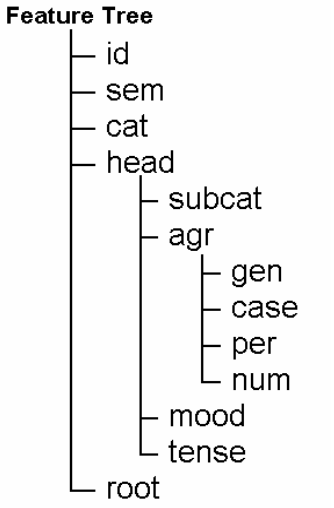


Figure 7. *Global Tree structure of the stored features.*

We consider in sequence the columns of Table 1, each of which represents a feature. Id (the foreign key) is necessary for making the record reachable from other sources within the ePOWER Workbench. The *sem* feature is the exact shape of the word as it occurs in the legal sentence. The *cat* feature states the grammatical category the word belongs to. The word used in the example has value *V*, which means that this is a verb. Other values are N, for nouns, PUNCT (lithographic punctuations like : ; ‘ ”] [( ) { } . ,), CUR is a value to indicate a currency symbol like €, PREP, for the indication of language prepositions and some more.

The *head.subcat* feature is a subdivision of the cat feature. For example the *subcat* feature value for the cat feature V is MAIN indicates that this verb is a main verb. Other values for the *subcat*-feature of a verb are AUX (*auxiliary*) and COP (*copula*). Values for the *subcat*-feature for the cat-feature NUM are ORD (*ordinal)* and CARD (*cardinal*). Other *cat* feature values and *subcat* feature values also exist, but at this moment I think the global idea is clear.

The example shows that *to deem* is indicated as the *root* of the main verb. This *root* feature information is used in the final translator. The next couple of features are sub features of the *head* feature. This feature is subdivided into four general sub features namely *subcat*, *agr*, *mood* and *tense*. The first was described earlier. The *agr* feature (for a description of how to use the grammatical agreement within Natural Language Parsing see The Natural Language Processing Dictionary 2003 [18]) is also subdivided into four sub-sub-features, namely *gen* (gender), *case* (e.g. object, subject), *per* (person), and *num* (number, e.g. singular and plural). These features give value to each of the grammatical agreements of the specific word in the legal sentence. The *mood* and the *tense* features of the *head* feature give a grammatical

meaning to verbs. E.g. values for the *mood* feature are INDICATIVE, INFINITIVE and PARTICIPLE. For the *tense* feature some values are PRESENT, PAST and IMPERFECT. All these *head* features are language dependent, so other values of the sub and sub-sub-features are possible. The *features* feature offers the possibility to extend the generic features with additional ones when necessary.

With all these grammatical information stored in one database table all necessary information becomes easily administrable and accessible.

6.2 *Unification Grammar*

In my approach (verb-phrase extraction), I also make use of the unification grammar[16] for parsing. This Unification Grammar is stored within the translate-nl database as a separate database table (ProductionRules table). Each record of this database is filled with information necessary for each of the specific Production rules (specified in XML, XML Developer Centre 2004 [19]). Each of these production rules declares some kind of grammatical part of the language used. Because, one of the building blocks of a Unification Grammar is Production rules we call this approach a rule based approach (see Rule Based Systems 1994 [20]).

In Figure 8, the general form of a Production rule is depicted.

LHS	**RHS**	
NLC	-> (NLC)+	**Grammar Rule**
(<Feature-Name> = <Feature-Value>)+		**Unification rules (Features)**

Figure 8. *General form of a Production rule.*

In general, a production rule consists of a Grammar rule (in EBNF) and one or more Unification rules. A grammar rule consists of a *left-hand-side* (LHS) and a *right-hand-side* (RHS)[17]. The RHS consist of one or more optional *Natural Language Constructs* (NLC's) and the LHS in general is the name of the production rule. Because the LHS is also a NLC, recursive grammars are possible. The Unification rules belonging to a Grammar rule each consist of a Feature-Name/Feature-Value pair.

[16] The Unification Grammar used within the Workbench consists of a set of Production rules. These production rules consist of one grammar rule, each consisting of one or more unification rules.

[17] When in this thesis the words "LHS"and "RHS" are used also the words "left-hand-side" and "right-hand-side" can be used.

An example of a production rule which can be used to parse determiners like *your*, *his*, *my*, *her*, *our* and so on is defined in Figure 9 (see De Determinator 2002 [21] for a detailed specification of grammatical determiners).

```
DETE -> PN

<DETE inResult>          = false
<PN head subcat>         = PERSONAL
<PN head agr case>       = C2
<DETE sem>               = <PN sem>
```

Figure 9. *The Grammar rule and the relevant Unification rules for the extraction of grammatical determiners.*

Figure 9 shows one of the production rules for the extraction of determiners. Also the Unification rules are specified. The first unification rule states that the result of this rule is not visible in the final result (i.e. can not act as a root of a parse tree). The second and the third rule are used to select only those PN's (pronouns) where the head.subcat *feature* is equal to PERSONAL and the head.agr.case *feature* is equal to C2 (genitive). All the feature information can be read from the lexicon, where all the (grammatical) features are stored (see Section 6.1). The last Unification rule states that the *sem*-feature of the DETE is equal to the *sem*-feature of the PN. With all these unification rules, some grammatical properties of the different NLC's can be stated and used in the parsing process.

All production rules are described in XML and stored in a separate database table, which makes the grammar easily to maintain and to access. When new grammatical functionality is necessary for the extraction of other language constructs one can specify new production rules and add them to the database.

The production rule depicted in Figure 9 is implemented during the first step of automated norm extraction (*Noun-Phrase Extraction,* also knows as concept extraction), because this grammatical element is always part of a noun-phrase. For the generation of the production rules, in the second step (*Verb-Phrase Extraction*), other choices have been made for storing the necessary parse information (in this case the information for all the different categorizations specified in Chapter 5). I have chosen to store all the necessary information of the different JLC's (categorizations) statically within each of the production rules[18] (see Figure 10), although I first considered using another possible solution method for this problem (see Chapter 11.1).

[18] This seems the most efficient way to implement the Production rules for each of the different JLC's, because this approach reuses all functionality, which was already present in the Workbench project when I started my thesis research.

S -> NP_1 V_2 (XLIST_3) V_4 XLIST_5

<S inResult>	= true
<S sem type>	**= s_dp**
<S s_order>	= sv
<V_2 root>	= worden
<V_2 head subcat>	= AUX
<V_4 root>	= achten
<V_4 head subcat>	= MAIN
<S head agr>	= <NP_1 head agr>
<S head agr>	= <V_2 head agr>
<S sem subject>	= <NP_1 sem>
<S sem dp_part1>	= <V_2 sem>
<S sem time_period>	= <XLIST_3 sem>
<S sem dp_part2>	= <V_4 sem>
<S sem fiction>	= <XLIST_5 sem>

Figure 10. *Static storage and usage of the JLC Deeming Provision (s_order = sv)*

Figure 10 shows that a special feature (sem.type = s_dp) is created to recognize this production rule as being a JLC Deeming Provision during the parsing process. When this rule is applicable to the legal sentence (input text), during the parsing process, this legal sentence is recognized as being of type "Deeming Provision". Later, during the translation part, this JLC information can be used to generate a translation pattern for this specific case. In the next section the complete translation process is specified.

In Chapter 12, some optimisations on the generation and management of the production rules will be discussed.

For now, let us look at the translation of the parsed information (by application of the Production rules) to a computational model (expressed in UML/OCL).

6.3 Translation Patterns

The last important part of the ePOWER Workbench is the database table with all the so called *translation patterns*. These patterns can be used to convert the result of the parsing process into a computational model. In case of the (E-)POWER approach these models are built upon the UML/OCL standards (see Fowler, M., Scott, K. 2000 [9] & Warmer, J., Kleppe, A. 1999 [10]). In the initial Workbench project, Visual Basic scripts are used to specify the conversion functionality (see Visual Basic Language and Run-Time Reference 2004 [22]).

An example of a pattern can be found in Table 2. This pattern uses information, which is saved during the application of the grammar rule *NP => CUR NUM* (see Figure 11).

Type = "np_money"
dim Result as New System.Collections.ArrayList dim Temp as Object Temp = Feature.Model.GetType(Feature.Item("cur").ToString + Feature.Item("root").ToString) Result.Add(Temp) return Result

Table 2. *The translation pattern for the type np_money.*

NP => CUR NUM

<NP inResult>	= true
<NP sem type>	**= np_money**
<NUM subcat>	= CARDINAL
<NP sem main>	= <NUM sem>
<NP sem root>	**= <NUM root>**
<NP sem cur>	**= <CUR sem>**

Figure 11. *The Production rule used to parse and store the information used in the np_money translation pattern to create the relevant computational model.*

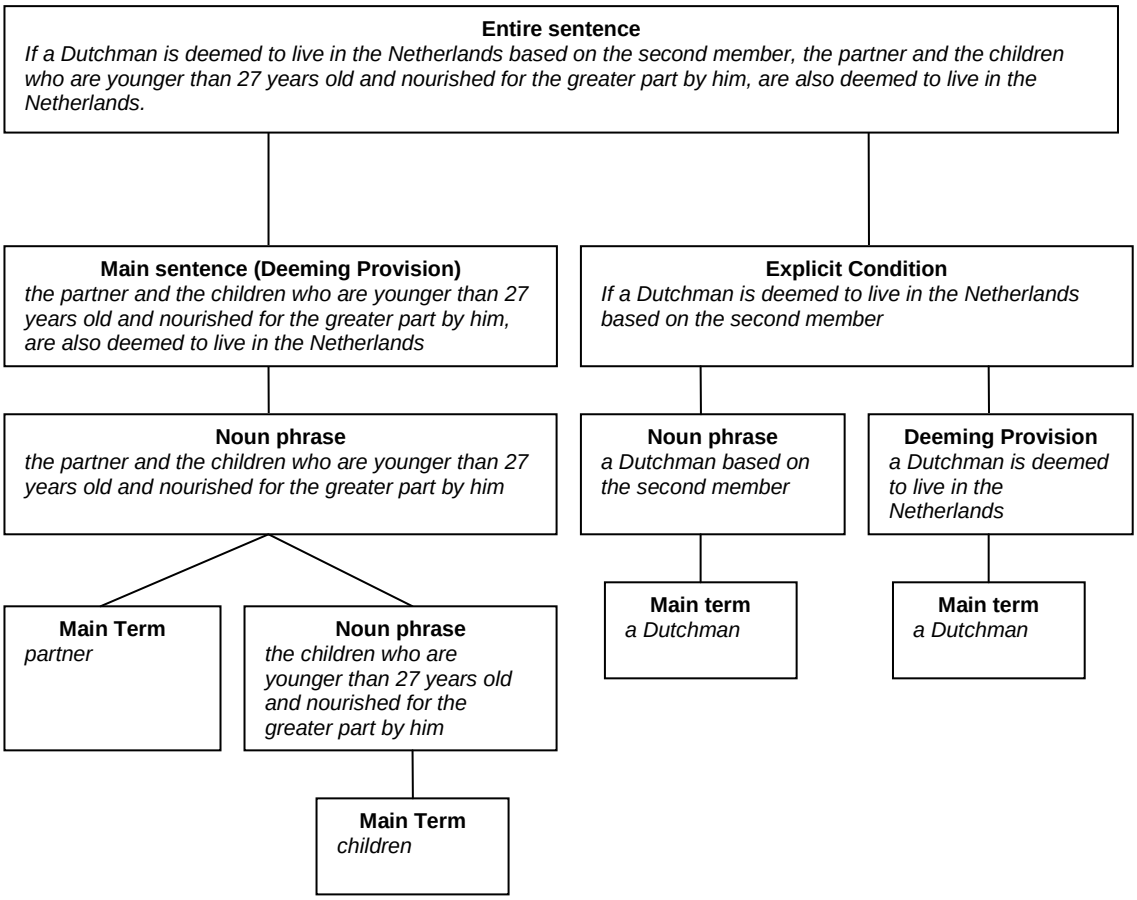
The translation pattern in table 2 shows that the variables *cur* and *root* are used to generate the relevant computational model. This is because these variables contain information, which has been stored during the application of the relevant Production rule. In Figure 11, the relevant Production rule is stated. In the RHS of this rule both the NLC's (CUR and NUM) are parsed and connected to the relevant *sem*-features in the LHS[19] (see Figure 11, the last two Unification rules). In general for each of the specified *sem.type*-features, declared in a Production rule in the LHS, a translation pattern (type = sem.type) must be made. By this fact, there is a possibility that more than one JLC can be recognized within the input text (legal sentence). Suppose we want to translate the following legal sentence:

IB2002 Article 2.2 Member 3
If a Dutchman is deemed to live in the Netherlands based on the second member, the partner and the children who are younger that 27 years old and nourished for the greater part by him, are also deemed to live in the Netherlands.

We first have to determine which categories can be found within this sentence. By a closer look, two main categories, namely an Explicit Condition and a Deeming Provision, can be found.

[19] In general the RHS NLC's, which has to be used in the final translation pattern or in a higher fragment of the derivation tree, are interlinked by the relevant LHS *sem*-features by *EquationId's*.

A global structure of the legal sentence shows us the existence of both JLC's:



For this legal sentence to be translated special translation patterns have to be made to handle the existence of multiple JLC's. There is a translation pattern made for the Deeming Provision (as a stand-alone JLC, see Appendix F "type = s_dp") and one for the Explicit Condition (as a stand-alone JLC, see Appendix F "type = ec") and some additional control statements within the translation patterns to handle the combination of both JLC's. In this case, special additional statements[20] are added within the translation pattern of the Deeming Provision to handle the existence of an Explicit Condition (see table 3).

Table 3 shows us which statements are necessary to handle the existence of an Explicit Condition in combination with a Deeming Provision. First we have to check if there is an Explicit Condition found during the parsing process (see table 3, line number 19). If there is an Explicit Condition then we collect all the attributes within this Explicit Condition (see table 3, line number 20/27) and add the condition statement to the condition part of the Deeming Provision (see table 3, rule 28/33).

[20] In this thesis research I have extended the ePOWER Workbench Translator (the Production Rule set) with functionality to handle a subset of all applicable JLC's and JLC combinations. The necessary functionality is added by examining all the legal sentences (from a self-made testbench, categorized by JLC-name) and their relevant JLC's. So the ePOWER Workbench contains only functionality to recognize and translate a subset of all possible JLC combinations. Therefore, this thesis can be seen as a first step for tackling the main problem of automated norm extraction from legal texts.

type = "s_dp"

```
1   dim Result as New System.Collections.ArrayList
2   dim EC as New System.Collections.ArrayList
3   dim Counter as Object
4   dim AttrCounter as Object
5   dim EcCounter as Object
6   dim Attr as Object
7   dim strCondition as String
8   dim strTemp as String

9   Result = Feature.Item("subject").Translate(Nothing)
10 for each Counter in Result
11   strCondition = ""
12   for each AttrCounter in Counter.myAttributes
13     if strCondition = "" then
14       strCondition = AttrCounter.Name
15     else
16       strCondition = strCondition + " and " + AttrCounter.Name
17     end if
18   next

19   if Feature.item("ec.type").ToString <> "" then
20     EC = Feature.Item("ec").Translate(Nothing)
21     for each EcCounter in EC
22       for each AttrCounter in EcCounter.myAttributes
23         if Counter.Name = EcCounter.Name then
24           strTemp = AttrCounter.Name
25         else
26           strTemp = EcCounter.Name + "." + AttrCounter.Name
27         end if
28         if strCondition = "" then
29           strCondition = strTemp
30         else
31           strCondition = strCondition + " and " + strTemp
32         end if
33       next
34     next
35   end if

36   if strCondition <> "" then
37     strCondition = strCondition + " implies "
38   end if

39   Attr = Counter.GetAttribute("Boolean", Feature.Item("dp_part1").ToString +
Feature.Item("time_period").Translate(Nothing) + Feature.Item("dp_part2").ToString +
Feature.Item("fiction").Translate(Nothing))

40   Counter.GetConstraint("attributeInvariant", strCondition + Attr.Name)
41 next

42 Return Result
```

Table 3. *The Translation Pattern for the JLC Deeming Provision; special statements to handle the existence of an Explicit Condition are marked bold.*

When this translation pattern, together with the translation pattern of the Explicit Condition, is added, the following computational model will be generated:

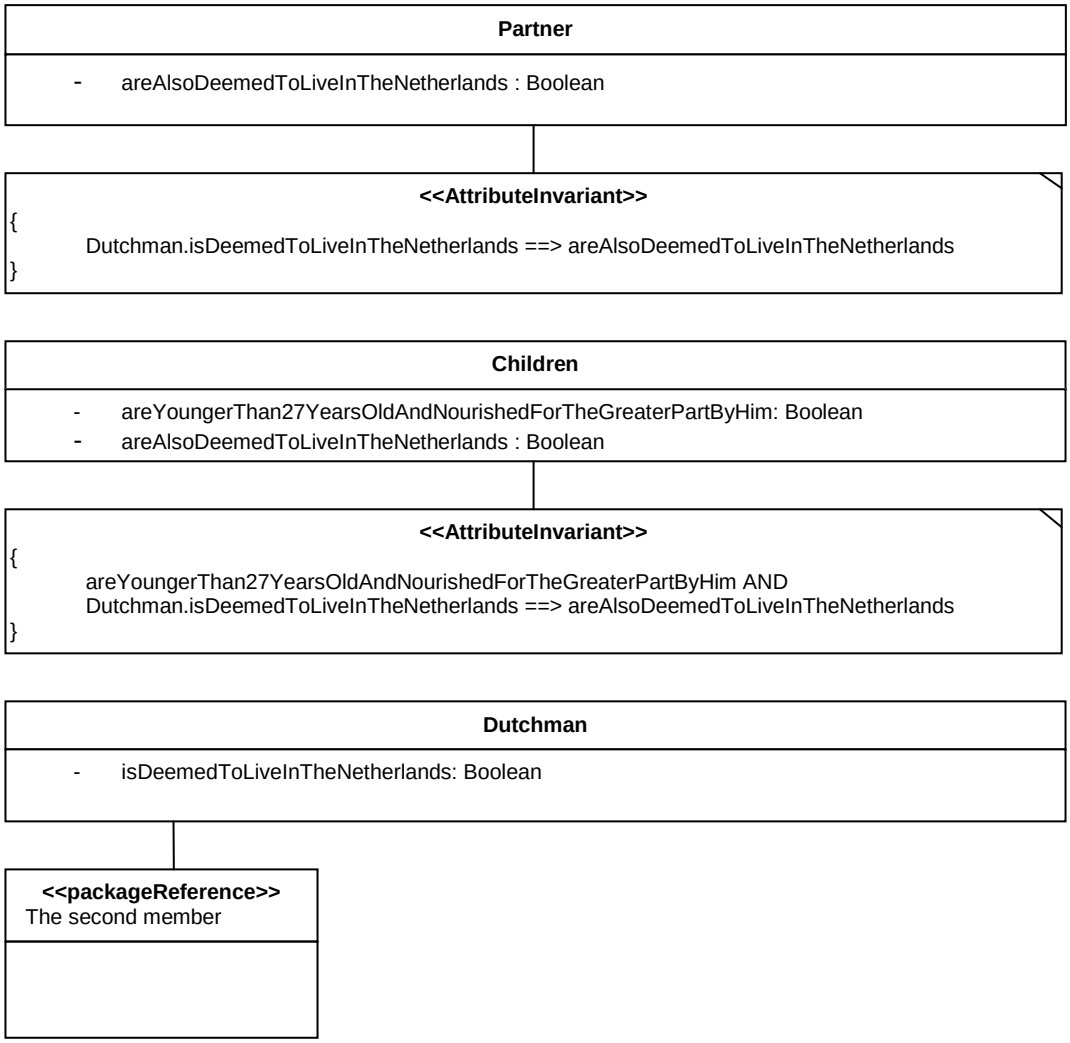


Figure 12. *The computational model of the law specified in IB2002 Article 2.2 Member 3*

So, when new functionality is added to the ePOWER Workbench (in other words adding new Production rules) new patterns have to be added or old ones have to be adjusted.

7 The Generation of Production rules

In this chapter, a description is given of how the different production rules can be generated by examining the different JLC's.

Specific for my approach is the treatment of the different JLC's. Figure 13 shows us the global structure of a subset of the rules that are used for parsing the JLC's. This model is not complete but can be extended if more knowledge about how to recognize and translate JCL's becomes available.

Sentence
(Sentence) S -> S (PUNCT) EC
head.s_order = SV
(Sentence) S -> EC PUNCT (ADV) S
head.s_order = VS
Optional JLC's
(Explicit Condition) EC -> [Indien] NP XLIST

Main Sentence JLC's
(Deeming Provision) S -> NP V1 XLIST V2 XLIST
sem.s_order = SV (V1)root = worden;(V2)root = achten
sem.type = dp
(Deeming Provision) S -> V1 NP (XLIST) V2 XLIST
sem.s_order = VS (V1)root = worden;(V2)root = achten
sem.type = dp
(Definition) S -> NP V XLIST
sem.s_order = VS (V)root = zijn
sem.type = def
(Definition) S -> V NP XLIST
sem.s_order = VS (V)root = zijn
sem.type = def

Figure 13. *The global structure of a subset of the rules that are used for parsing the JLC's*

The above structure is divided into three parts corresponding to the general structure of a legal sentence (see Figure 4). In this figure, a normative Sentence is described consisting Optional JLC's and Main Sentence JLC's. In this global structure, all optional JLC's have a corresponding grammar rule and are part of a sentence.

Furthermore, a division is made in the order of the subject and verb constructs. There are (legal) sentences with the subject followed by some kind of verb (normal sentences) and other (legal) sentences with some verb followed by the subject (subordinate clause). Because every sentence can contain a subordinate clause, for every Main Sentence JLC two or more Grammar rules must be made.

The parsing engine can handle recursive rules. This is necessary because legal sentences can contain more than one JLC. A JLC can consist of another JLC and so on and so on.

In general, when a new JLC has been specified (by further research) one can look at the general functionality of the JLC (optional or not optional) and add the relevant information to the model. When the new JLC is optional, we have to add two rules

(SV/VS-form) to the Sentence part of the foregoing structure and one rule to the Optional part. When it is not optional, we have to add two rules (SV/VS-form) to the Main Sentence JLC's part.

8 An example; parsing a JLC

In this chapter, we show how we can apply the different techniques described in Chapters 5, 6 and 7 to obtain, for the example JLC Definition 1, the production rule (a grammar rule and unification rules) and the translation pattern. The former is used to recognize sentences of the given category, the latter to generate the corresponding formal model. Note that what we describe here for this specific JLC, has been done for each of the known JLC's listed in Appendix A. The outcome of this process can be found in Appendix B and E.

An example of a JLC is the Definition 1. The basic format of the JLC Definition 1 is (see Appendix A):

<subject> [are | is] <definition> (1)

When we want to translate the following legal text

IB 2001 Art 2.1 section 2
Dutch income is income as meant in chapter 7.

a couple of subsequent actions have to be made. First, we have to determine a production rule by examining the global structure of the JLC Definition 1. After we have recognized a production rule, we can determine a translation pattern for the generation of the relevant formal model. First, let us take a closer look at the production rule of the JLC Definition 1 (see Figure 14).

S -> NP_1 V_2 NP_3

1.	<S inResult>	= true
2.	<V_2 root>	= be
3.	<V_2 head subcat>	= MAIN
4.	<NP_3 sem isValue>	= false
5.	<S head agr>	= <NP_1 head agr>
6.	<S head agr>	= <V_2 head agr>
7.	<S head>	= <V_2 head>
8.	<S sem subject>	= <NP_1 sem>
9.	<S sem direct_object>	= <NP_3 sem>
10.	<S sem type>	= s_def

Figure 14. *The Production rule for the JLC Definition 1*

The above figure presents us the production rule for the JLC Definition 1. This production rule is made by examining the global structure of the JLC Definition 1 (see (1)). There are three NLC's made, namely NP_1 (for the extraction of the subject), V_2 (for the recognition of the main term "is") and NP_3 (for the recognition of the description of the definition part of the legal sentence). In addition, a couple of unification rules are made to be able to enforce some restrictions during recognition part of the parsing process. I will discuss the unification rules one by one so the global meaning of the complete production rule will become clear.

The first unification rule can be used to depict the outcome of this rule in the final derivation tree. When the value for the variable inResult is set to false, the outcome of the parsing process is not depicted in the final parse tree. Mostly those rules (with inResult = false) are part of a higher-level rule for the generation of sub elements of the higher level construct. In our case, we do not need the result of this rule to be visualized in the final answer. In general, when a legal sentence is classified as being of JLC type Definition 1, we have to generate two classes (one for the first NP and one for the second NP (the right-hand-side of the global structure of the JLC Definition 1)) whereby the first class is a sub class of the second class (see Figure 15).

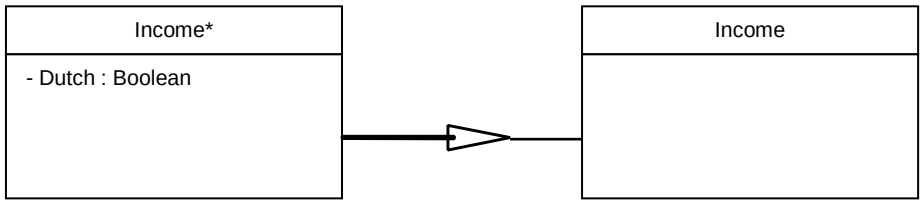


Figure 15. *Income* is a sub class of the class Income (Inheritance).*

When we look at the above figure we note that in our case we have to make a class "Income*" and a class "Income". See Section 10.2 for a detailed description of how to handle inheritance.

Both NP's are used in a higher level of the parse tree for the recognition of other formal statements within each NP. So the result of this rule can be seen as an intermediate result and thereby it is not visible in the final result.

The rules 2 and 3 (in Figure 14) are used to make some grammatical restrictions on the recognized verb. Both rules will enforce that we recognize the word "is", namely by specifying that the root-feature is equal to *to be* and the head.subcat-feature is equal to *main*. Other grammatical forms of the verb *to be* are possible, but in the example only the verb *is* has to be recognized.

Unification rule 4 is a special rule for the recognition of values (words which always represent values, like "height", "level", "amount", "10" etcetera). In this case, for the classification of the JLC Definition 1, we only want to recognize non-value language constructs on the right-hand-side of the JLC Definition 1. When the right-hand-side NP can be seen as a value-statement, we can apply the production rule for the JLC Value Assignment, Change and Comparison (see Appendix A). See Section 10.1 for a more detailed description of how to take care of language constructs, which represent values.

The unification rules 5, 6 and 7 make some restrictions on the grammatical agreement of the first NP (NP_1) and the verb (V_2). Verb (V_2) must agree with its subject (NP_1) in person and number (see The Natural Language Processing Dictionary 2003 [18]).

The next two unification rules, 8 and 9, are used to store the recognized language constructs into variables, so during the generation of the relevant formal model we can use these variables. Note that we only need the first NP (as subject) and the second NP (as direct_object) for the generation of the formal model. The reason that we do parse and store the other natural language constructs is the fact that in the

future we have to generate OCL constraints to be able to generate code (see Liduan, F. 2004 [31]). At this moment, we just want to visualize the result of the translation process (at this moment by using the UML/OCL conventions).

The last unification rule is used to indicate that this rule is of JLC type Definition 1 (s_def). Later, when we want to implement a translation pattern for this production rule we can reach the parsed information by referring to the stored variables (subject and direct_object).

After determining of the production rule, we can specify a relevant translation pattern. Table 4, shows us such a translation pattern, which can be used to generate the formal model of the JLC Definition 1.

type = "s_def"

dim Result as New System.Collections.ArrayList
dim Temp as New System.Collections.ArrayList
dim Counter as Object
dim Counter2 as Object
dim Assoc as Object
dim strConstraint as String
dim boolFound as Boolean
 	
strConstraint = ""
 	
Result = Feature.Item("**subject**").Translate(Nothing)
for each Counter in Result
  Counter.Name = Counter.Name + "*"
next
Temp = Feature.Item("**direct_object**").Translate(Nothing)
for each Counter in Result
  boolFound = false
  for each Counter2 in Temp
  if Counter.Name = Counter2.Name + "*" then
  boolFound = true
  end if
  next
  if boolFound = false then
  Counter.Name = Left(Counter.Name, Len(Counter.Name) - 1)
  end if
next
 	
for each Counter in Temp
  for each Assoc in Counter.myAttributes
  strConstraint = strConstraint + " and " + Assoc.Name
  next
  for each Assoc in Counter.myAssociations
  strConstraint = strConstraint + " and " + Assoc.Name + "->notEmpty"
  next
  if strConstraint <> "" then
  strConstraint = strConstraint.SubString(5)
  end if
next
 	
for each Counter in Result
  Counter.Supertype = Temp(0)
  if strConstraint <> "" then
  Counter.GetConstraint("Invariant", strConstraint)
  end if
next
 	
Result.Add(Temp)
return Result

Table 4. *The translation pattern of the JLC Definition 1, with the parsed information marked bold. Note that a Constraint is built and stored as a String.*

It shows us that both the NP's (subject and direct_object) parsed by application of the production rule are used to generate the relevant model. I will not explain the complete meaning of each of the subsequent rules within the translation pattern. I suppose that the reader will understand this script. Finally, the complete formal model generated by application of the production rule and the translation pattern is depicted in Figure 16.

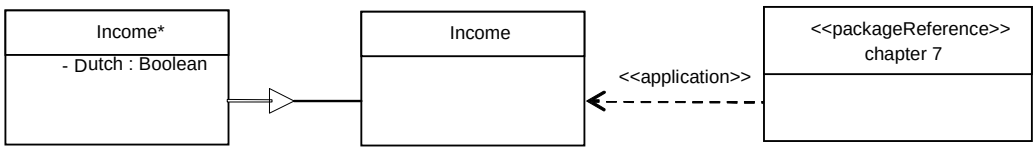


Figure 16. *UML/OCL model describing the definition as expressed in "Dutch income is income as meant in chapter 7".*

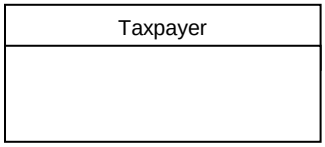
By looking at the figure above, it becomes clear that from the legal sentence (depicted at the beginning of this chapter) both the subject- and definition part are translated to their corresponding (UML) classes. In this case, the definition includes a third class: a package reference. This has been created from another JLC, but this will not be discussed here.

9 Class-Attribute Generation

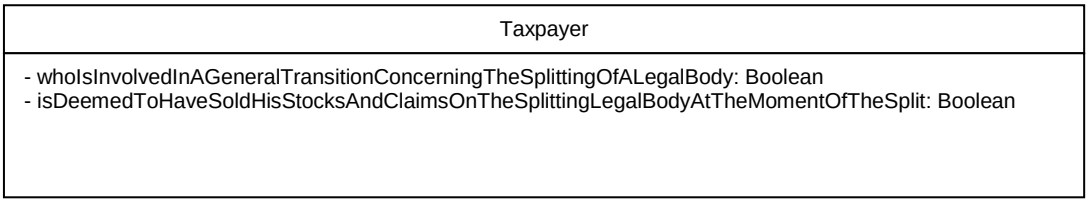
In previous chapters, top level parsing is used to recognize the relevant (natural) language constructs (JLC's). So, only relevant nouns (grammatical necessity) within the legal sentence are recognized as being part of a noun-phrase. Other nouns (like nouns within a subordinate clause) are often recognized as being part of an attribute. For example, when we want to translate the following legal sentence

IB 2001 Art 3.56 lid 1
The taxpayer who is involved in a general transition concerning the splitting of a legal body is deemed to have sold his stocks and claims on the splitting legal body at the moment of the split.

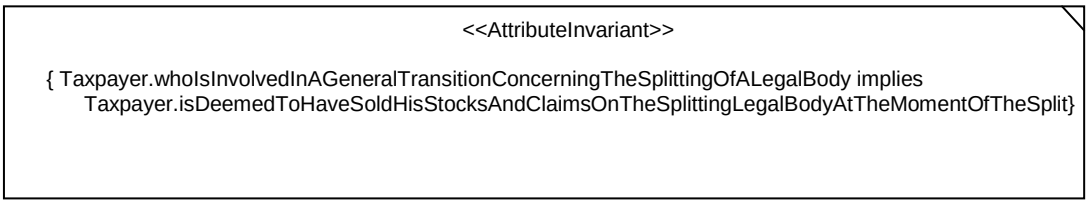
the subject (one of the main terms of the JLC Deeming Provision, see Appendix A and F "type=s_dp") of the sentence is "the Taxpayer", so a class-type Taxpayer is constructed.



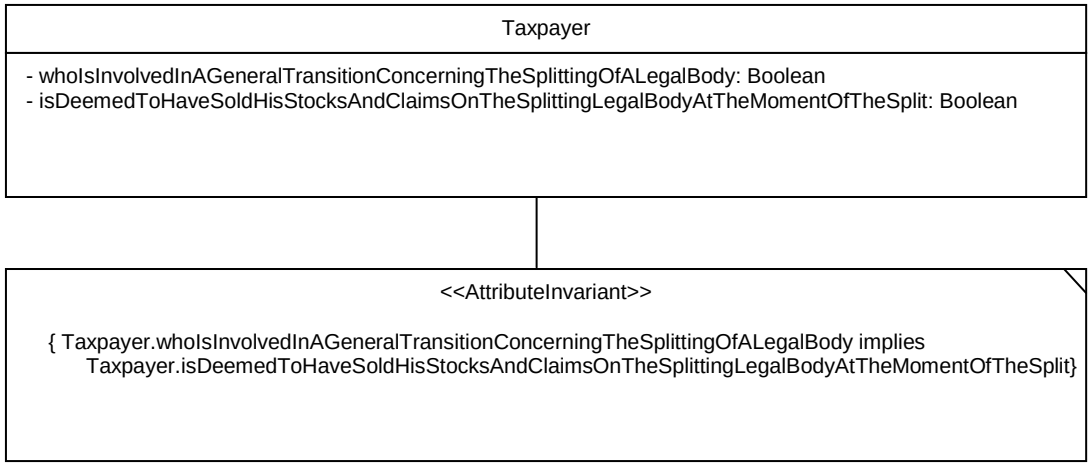
Further, we can see that in the subordinate clause we have two subsequent attributes, which have to be made. Both are added to the class Taxpayer.



Also an "*AttributeInvariant*" has to be made according to the translation pattern (described in Appendix F "type=s_dp") which is built upon these both attributes.



The complete formal model is as follows:



In the previous formal model, we saw that many complicated formal attributes are constructed. For the generation of these formal attributes special functionality has been added to the ePOWER Workbench. The production rule of the *Deeming Provision* (the JLC that is recognized in the foregoing example) can be found in Figure 17.

S -> NP_1 V_2 (**XLIST_3**) V_4 **XLIST_5**

<S inResult>	= true
<S sem type>	= s_dp
<S s_order>	= sv
<V_2 root>	= worden
<V_2 head subcat>	= AUX
<V_4 root>	= achten
<V_4 head subcat>	= MAIN
<S head agr>	= <NP_1 head agr>
<S head agr>	= <V_2 head agr>
<S sem subject>	= <NP_1 sem>
<S sem dp_part1>	= <V_2 sem>
<S sem time_period>	= <**XLIST_3** sem>
<S sem dp_part2>	= <V_4 sem>
<S sem fiction>	= <**XLIST_5** sem>

Figure 17. *The Grammar rule and Unification rules of the Deeming Provision JLC (the X-LIST NLC's are marked bold).*

Figure 17 shows that for the recognition/generation of attributes a special production rule[21] should be added to the production rule set, namely the *x-list* rule (see Appendix C "Production Rules for the Verb-phrase Extraction" and F "type=x_list"). The goal of this production rule is the possibility to generate attribute-names. From Figure 17 it becomes clear that the complete sentence part (language construct) between and after the subsequent language constructs NP_1 (noun-

[21] This rule is specified within the Grammar rule set (set of production rules), but it isn't really a rule specified for grammatical purpose. This rule is added for making it possible to concatenate different language constructs (for the purpose of attribute generation).

phrase) V_2 (verb) and V_4 can be seen as one single attribute (X-LIST). This sentence part can of course consist of a single word or a word-phrase, so the x-list production rule has to handle a list of words. When we look at the specification of the production rule x-list (see Figure 18) we can see that every word (except for all the punctuation marks (cat-feature is PUNCT)), read from the lexicon must have the feature value cat = X (for the generalisation of every word being an X).

XLIST -> X_1 (XLIST_2)	
<XLIST inResult>	= false
<XLIST sem type>	= x_list
<XLIST sem hd>	= <X_1 sem>
<XLIST sem tl>	= <XLIST_2 sem>

Figure 18. *The Production rule for the X-LIST*

When every word is of type X we can use it as input for the x-list production rule, so it can be transformed to a single attribute. Initially, this functionality wasn't specified within the ePOWER Workbench, so some adaptations were necessary (see Appendix B) (in Microsoft .NET 2003 [23] you can find the documentation of Microsoft .NET). With this adaptation of the programming code, we now have the possibility to generate attribute names. The generation/addition of the attributes is done during the translation step of every JLC. When an attribute can be generated from a sub construct, specified within a JLC, this can be done by application of the x-list production rule.

A second reason for introducing such a x_list production rule is the ability to parse natural language construct for which we do not have the legal knowledge yet. When there is no such rule available (recognizing a sequence of words) the recognition step will not successfully finish, because we cannot recognize the complete legal sentence. The x_list production rule is a generic construction to deal with pieces of text for which we have no information or have no knowledge about its internal representation. When we introduce such a rule we are able to incrementally implement a tool for automated norm extraction. Every time when new legal knowledge becomes available, we can add it to the relevant JLC or add some new JLC's.

A disadvantage of such a production rule is the fact that the translation process becomes ambiguous. When such a rule is applicable, a sequence of words has been recognized. In some cases for every member of this sequence a different derivation tree will generated. This because during the parsing process it is not possible to determine the end of the rule. The next word or word phrase can be of another JLC (or NLC) but can also belong to the x_list JLC itself.

10 Special Treatment

During my thesis research there were some choices made about the treatment of special language constructs found during examination of the different legal sentence. Some thought gives rise to some adaptations of the relevant production rules, translation patterns or other ePOWER Workbench parts (adding extra special features to the relevant production rule, adding some special code fragments in the relevant translation pattern, or adding some special records in the lexicon) to optimise the extraction of the special language constructs.

In the first section, I will discuss how I differentiate between the JLC Definition 1 and the JLC Value Assignment, Change and Comparison. Those JLC's are similar by way of their similar global structure (see Appendix A). To make a distinction between the global meaning of both categorizations some special care was necessary.

In the following section, a description is given about the choices made during the modelling of inheritance. Moreover, the final section describes the special treatment of fixed noun-phrases.

10.1 Values

During the implementation of the JLC's Definition 1 and Value Assignment, Change and Comparison some special treatment was necessary. Because both global structures are very similar, we have to make some special choices to be able to distinguish between both JLC's. First, let us look at the global structure of the JLC Definition 1:

<subject>**[are|is]**<definition>

In general, this global structure is used to recognize inheritance from a legal sentence. This means that we have a special relation between the two class types. The first class is specified in the first noun-phrase (subject part) and the second class can be extracted from the last noun-phrase (definition part). The relation between both classes is that the first class is a subclass of the second class. This means that the first class inherits all formal attributes and methods from the second class. Within the first class, also some extra attributes and methods can be specified. By this fact, we can suppose that the second noun-phrase always is in the form of a class type with some additional formal elements (like attributes, subtypes, associations and constraints).

The global structure of the JLC Value Assignment, Change and Comparison is as follows:

<subject>**[is|amounts to]**<formula>

This structure can be used to recognize and extract value assignments. Hereby, the second noun-phrase is always in the form of a value (a property which can have a value or can be measured). This fact distinguishes the two JLC's (the meaning of the second noun-phrase).

To handle the difference in both global structures we have to make some efficient choices. My choice was to determine during the parsing process if the language construct to be parsed can be viewed as a value or not. When it is in the form of a value, like “amount”, “compensation”, “wage”, “salary”, “taxes” or “interest” we can conclude that application of the JLC Assignment, Change and Comparison is necessary. Otherwise, the JLC Definition 1 is applied.
By adding a special feature to the relevant words in our lexicon we can during the parsing process determine which JLC is applicable. Table 5 shows us an example of two records in the lexicon with the extra feature (in the form of an attribute-value-pair “isValue=true/false”) used for the value determination step.

Lexicon											
id	**sem**	**cat**	**head .sub cat**	**root**	**head .agr. gen**	**head .agr. case**	**head. mood**	**head .tens e**	**head .agr. per**	**head .agr. num**	**Features**
4	amount	N		amount	N					S	**isValue**="true"
4	article	N		article	N					P	**isValue**="false"

Table 5. *Two records from the lexicon with the extra feature necessary to determine if this word can be characterized as a value.*

When we look at the production rules for both JLC’s (see Figure 19 and 20) the specific `Sem.isValue` feature is used to handle the determination of the correct language construct.

```
S -> NP_1 V_2 NP_3

<S inResult>              = true
<S sem type>              = s_def
<V_2 root>                = zijn
<V_2 head subcat>         = MAIN
<NP_3 sem isValue>        = false
<S head agr>              = <NP_1 head agr>
<S head agr>              = <V_2 head agr>
<S head>                  = <V_2 head>
<S sem subject>           = <NP_1 sem>
<S sem direct_object>     = <NP_3 sem>
```

Figure 19. *The production rule for the JLC Definition 1, with the relevant unification rule for the determination of a value statement marked bold.*

S -> NP_1 V_2 (N_3) (PREP_4) NP_5

<S inResult>	= true
<S sem type>	= s_va
<NP_1 sem isValue>	**= true**
<V_2 root>	= {zijn,bedragen}
<V_2 head subcat>	= MAIN
<N_3 root>	= gelijk
<PREP_4 root>	= aan
<NP_5 sem isValue>	**= true**
<S sem subject>	= <NP_1 sem>
<S sem formula>	= <NP_1 head agr>
<S sem formula>	= <V_2 head agr>
<S sem formula>	= <NP_5 sem>

Figure 20. *The production rule for the JLC Assignment, Change and Comparison, with the relevant unification rule for the determination of a value statement marked bold.*

When by referencing to the lexicon it appears that the language construct has the property that it characterizes a value (so, "isValue=true") then the JLC Assignment, Change and Comparison is applicable. Otherwise, the JLC Definition1 applies. Our initial problem of distinguishing both JLC's is solved.

Sometimes it is not possible to determine if some word characterizes a value. For example, when we want to translate the following two legal sentences

IB2001 Article 2.1 member 1
*Dutch **income** is income as meant in chapter 7.*

IB2001 Article 3.3 member 1
*Taxable **income** is income reduced with the employee's discount.*

we are not able to distinguish both legal sentences by looking at the extra feature (isValue) described for the word "income". This because in the first legal sentence the word "income" is not used as a value (so the JLC Definition 1 is applicable) and in the second legal sentence the word is used as a value (so the JLC Assignment, Change and Comparison is applicable). We cannot add the extra feature *isValue* to the word "income" because this word can be used in both ways. In such cases the user should choose which JLC is applicable (both derivation trees are generated during the translation step).

10.2 Inheritance and the Generation of Class Names

In this section, a description is given on how I have handled the occurrence of inheritance within a legal sentence. In previous sections there is mentioned that at this moment within the ePOWER Workbench the production rule for the JLC Definition 1 can be used for the recognition of inheritance occurring in a legal sentence. For the generation of the relevant formal model the translation pattern of the JLC Definition 1 can be found in Table 6.

type = "s_def"

dim Result as New System.Collections.ArrayList
dim Temp as New System.Collections.ArrayList
dim Counter as Object
dim Counter2 as Object
dim Assoc as Object
dim strConstraint as String
dim boolFound as Boolean
 	
strConstraint = ""
 	
*Result = Feature.Item("**subject**").Translate(Nothing)*
for each Counter in Result
Counter.Name = Counter.Name + ""*
next
*Temp = Feature.Item("**direct_object**").Translate(Nothing)*
for each Counter in Result
boolFound = false
for each Counter2 in Temp
if Counter.Name = Counter2.Name + "" then*
boolFound = true
end if
next
if boolFound = false then
Counter.Name = Left(Counter.Name, Len(Counter.Name) - 1)
end if
next
 	
for each Counter in Temp
for each Assoc in Counter.myAttributes
strConstraint = strConstraint + " and " + Assoc.Name
next
for each Assoc in Counter.myAssociations
strConstraint = strConstraint + " and " + Assoc.Name + "->notEmpty"
next
if strConstraint <> "" then
strConstraint = strConstraint.SubString(5)
end if
next
 	
for each Counter in Result
Counter.Supertype = Temp(0)
if strConstraint <> "" then
Counter.GetConstraint("Invariant", strConstraint)
end if
next
 	
Result.Add(Temp)
return Result

Table 6. *The translation pattern of the JLC Definition 1 with the relevant code for handling inheritance marked in italics.*

Table 6 shows us the translation pattern of the JLC Definition 1 with the relevant code for handling inheritance marked in italics. In the specific code fragment you see that we first translate the *subject* construct (the first noun-phrase specified in the global structure of the JLC Definition 1, see Appendix A). This will result in the generation of a collection of classes, namely *Result*. In addition, the second noun-phrase (*direct_object*) specified in the global structure is translated. This will also result in a collection of classes, namely *Temp*. After we have translated both constructs, we have to add some more information necessary for modelling inheritance. First, we traverse the complete set of classes specified in the Result collection and add a star (*) to each of the class names (*Counter.Name + "*"*). In the next couple of rules in the code we traverse the other collection and check if there is a class name in the second set which is equal to a class name in the first set. If so,

then we leave the Counter.Name as it is, and otherwise we cut off the star (*) from the Conter.Name. This process is necessary to be able to model inheritance when the class names of the super class and the sub class are the same. When this is the case we have to add some extra information to visualize this difference in the model. When the class names of the superclass and the subclass are different, the generation of the class names is delegated to the translation pattern of the noun-phrases. This seems a temporary solution, but at this moment this solutions is the best I have found during my research. In Figure 21, a legal sentence is depicted whereby the class name of the super class is equal to the class name of the sub class. Also the final model is depicted, so one can understand in which form inheritance is modelled by using our current translation engine.

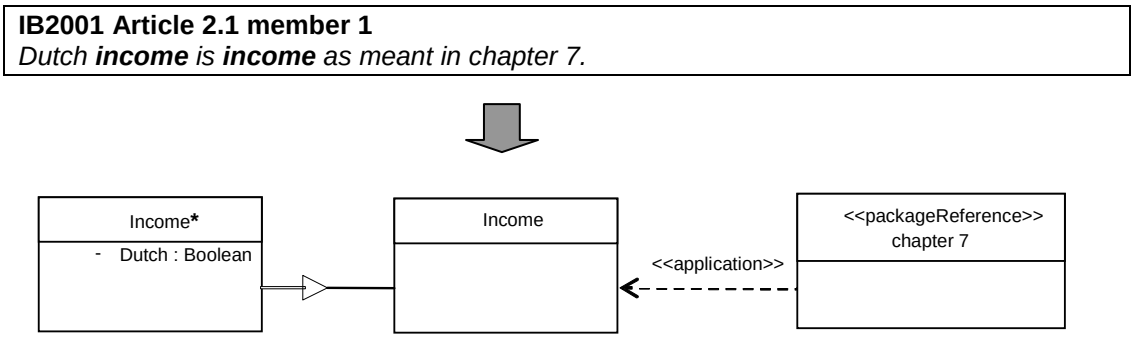


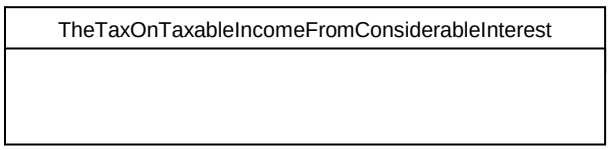
Figure 21. *The translation of a legal sentence containing inheritance (where the class name of the super class is equal to the class name of the sub class).*

During the complete implementation of the ePOWER Workbench, some choices for correctly modelling inheritance have been made. The simplest form of handling inheritance was the generation of class names by concatenating all subsequent language constructs from which the complete noun-phrase consists of (for example "Dutch Income", "A 24 years old Dutch student"). This is a very impractical choice because the final class names will have a lot of overhead and the formal elements within the noun-phrase are not properly recognized and translated.

For example, when we want to generate a class name from the following noun-phrase

The tax on taxable income from considerable interest…

the resulting formal model for this class after translation will be as follows:



When we look at the above figure we see that there are no formal elements added to the class, because we simply have not recognized them (initially there was no knowledge about the global (internal) structure of noun-phrases). We use the complete noun-phrase directly as being a class name.

Later on some functionality was added to let the user choose the best possible way of assigning a class name. This was possible because more knowledge becomes available about the global structure of noun-phrases. Special production rules were made for the recognition of complete noun-phrases, so more information becomes available during the final translation part (the generation of the formal models). During the translation part, the user is able to point out the relevant information necessary to construct the class name (for example the user can choose to concatenate the whole noun-phrase (old version), or he can choose to concatenate the adverb/preposition with the main term etcetera). This seems a better solution, but at this moment, this functionality isn't available anymore within the ePOWER Workbench.

So, in my thesis research I have tried to handle the occurrence of inheritance and the generation of class names by specifying new production rules and translation patterns as described in the beginning of this section. This seems the best solution possible with the current knowledge available for noun-phrases. Maybe in future development a more efficient and transparent solution can be found for generating class names and therewith the modelling of inheritance.

10.3 Fixed Noun Phrases

In Chapter 6 it was mentioned that all the words relevant for the Dutch language are stored in the so called lexicon database table in the translate-nl database. When, during the translation process, a production rule is applicable, for each subsequent word in the legal sentence (input), a reference to this lexicon is made.

In my thesis research, I found some word-phrases (noun-phrases) which always have a fixed structure when recognized in a legal sentence. One can think of the word-phrases, "The Dutch Kingdom", "Law on Income Taxes 1964" etcetera. These noun-phrases are always present in this form in a legal sentence. Therefore, it seems a good idea to store these complete noun-phrases as a whole in the lexicon database table. Otherwise, special production rules have to be made to recognize these specific noun-phrases. This seems more trouble than it is worth.

At this moment only those special production rules are made for the recognition of these fixed noun-phrases (for making it possible to recognize the legal sentences described in the testbench). Figure 22 shows us the production rule for the recognition of the fixed noun-phrase "Koninkrijk der Nederlanden" [22].

[22] The English word for the Dutch word "Koninkrijk der Nederlanden" is "The Dutch Kingdom".

NP -> NP_1 NP_2
<NP inResult> = true
<NP sem root> = Koninkrijk der Nederlanden
<NP sem type> = np
<NP_1 sem root> = koninkrijk
<NP_2 sem root> = Nederland
<NP_2 head agr case> = C2

Figure 22. *The production rule for the recognition and translation of the fixed noun-phrase "Koningrijk der Nederlanden".*

Also, other production rules are made for the recognition of other fixed nou-phrases, but at this moment I think the usability of those fixed noun-phrases is clear.

Maybe in the future there will be some time to examine the legal sentences, described in the legislation, for the occurrence of those fixed noun-phrases. This will result in a set of fixed language constructs. Herewith we will have all the knowledge necessary to extend the lexicon with the set of fixed noun-phrases. During the parsing process, these specific language constructs can be directly extracted from the lexicon (as a noun). So the relevant formal elements (class name or attribute) can simply be generated.

11 An Alternative Solution

In this chapter, an alternative solution to the main problem will be discussed. This solution was found during the preliminary research (determining a global approach for Automated Norm Extraction). All possible solutions found were considered, but are not implemented, because there were some shortcomings in implementation or efficiency. These shortcomings will be discussed in this chapter so it will become clear why these approaches have not been used. Maybe some parts of these approaches can be used in future optimisations or researches.

11.1 Dynamic JLC storage

In the previous chapters, a global solution to the main problem is discussed. This approach makes use of the categorization of the legal sentences (see Chapter 5). By these categorizations, subsequent language constructs are (also called as the Natural Language Constructs, NLC's, see Section 6.2) specified. These are, within the translation engine, used to determine the necessary normative constructs for the generation of the formal model. Because this process uses these subsequent structures (in the ePOWER Workbench stored as Production rules) for the parsing process, this approach is called a rule based approach (see Rule Based Systems 1994 [20]). In this approach, the JLC information of each of the categorizations is statically used within each of the composed Production rules. For every JLC one or more production rules have been made by examining the global rule based structure of the JLC's (see Appendix A and C for all the production rules) and by following the regulations of the rule model (see Figure 13 in Chapter 7). This all seems an efficient way of tackling the main problem, but the question arises if this can be done in a more efficient way.

By examining the global structure of the different JLC's, it may be noted that we mainly recognize the *main terms* of the JLC's to classify the legal sentence to one or more specific JLC's (see Figure 23 and Appendix A).

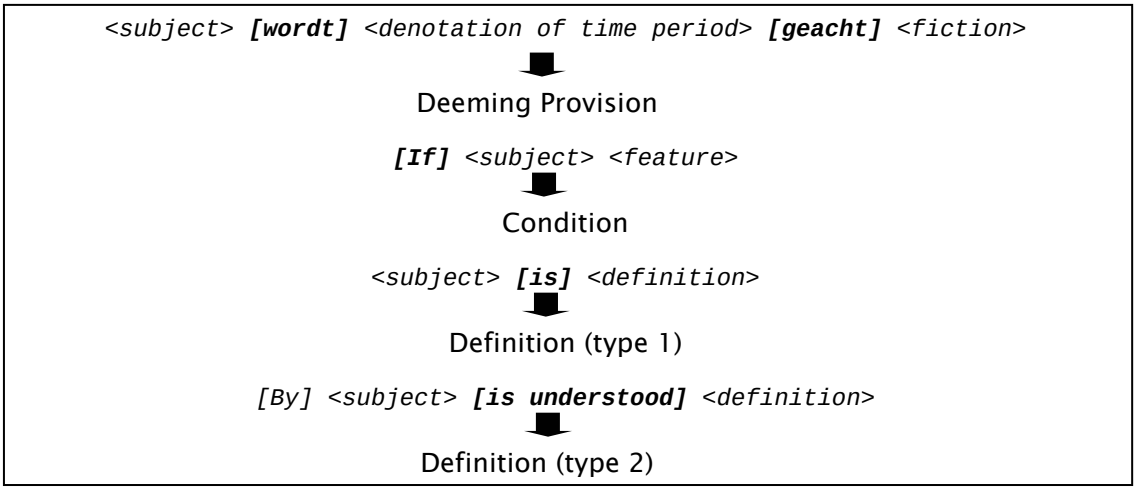


Figure 23. *The global structure of a subset of all possible JLC's, with the main terms marked bold.*

By this fact we can introduce another approach, which is based upon a Unification Algorithm and dynamic storage of the main terms of the global structure (in a database) of each of the different JLC's (so, this approach doesn't make use of the rule based structure of the different JLC's like in the previous approach). Dynamic storage of the JLC dependent language constructs also has the advantage that the JLC information is easy adaptable in future development. So the JLC information becomes easy administrable. When new JLC's are determined (and have to be added) or current ones have to be modified these information is easy modifiable in the ePOWER Workbench translation engine.

The dynamic storage of the different main terms can be done by adding the JLC type information to each of the relevant words stored in the lexicon. This is based upon the fact that we can add extra user defined features to each word in the lexicon table (see Table 8 and Section 6.1).

Lexicon											
Id	**sem**	**cat**	**subcat**	**root**	**Gen**	**Case**	**Mood**	**Tense**	**per**	**num**	**Features**
6	**deemed**	V	MAIN	**deem**			INDICATIVE	IMPERFECT	2	P	**JLCtype = DP**

Table 8. *Dynamic storage of the JLC type information in the lexicon as extra feature*

Table 8 shows us the possibility to store the JLC dependent type information for the Deeming Provision JLC as attribute-value-pair to the Features column. Every main term of the different JLC's will be extended with this JLC type information in the lexicon. In addition, words that are not part of the global structure of a JLC must have this attribute-value-pair as extra feature within the specific column. So it becomes possible to determine the JLC type information for each word during the unification process.

By examining the global structures of the different JLC's we found that one specific word (like "is") is used within more than one JLC specification. So it seems necessary to be able to store more than one JLC type by each word in the lexicon, as in Table 9.

Lexicon											
id	**sem**	**cat**	**subcat**	**root**	**gen**	**case**	**mood**	**Tense**	**per**	**num**	**Features**
6	**?**	V	MAIN	**?**			INDICATIVE	IMPERFECT	2	P	**JLCtype = {DP,TypeE}**

Table 9. *Dynamic storage of the JLC type information when more than one classification is possible*

The value of the attribute -value-pair contains all possible JLC types where a specific word classifies to.

During the parsing process, the final JLC type will be determined by examining all words (by continuously referencing to the lexicon for each word) and unifying to the specific JLC type(s). This unification algorithm will determine the final JLC type by taking the intersection of all the possible different sets (attribute-value-pairs) of JLC types of each parsed word. For example, when after the parsing process we have recognized three main terms (words with necessary JLC information) like

"by"	JLCtype={DEF2}
"is"	JLCtype={DEF1,DEF2}
"understood"	JLCtype={DEF2}

the complete classification process will result in the fact that the legal sentence is classified as a Definition 2, because the intersection of {DEF2}, {DEF1,DEF2} and {DEF2} is {DEF2} (see Appendix A). In principle, the sentence could be classified to the JLC Definition 1, but the presence of other normative elements ("By" and "understood") tells us that the classification as JLC Definition 2 is the most likely one.

11.1.1 Excluding Invalid NLC Sequences

In the previous section an approach was discussed which makes use of a unification algorithm for the classification of the different legal sentences to one or more JLC types. Because this approach is not rule based, every combination of NLC's is possible (as long as they have the same relevant attribute-value-pair in their feature column in the lexicon database table). For example by the above approach a legal sentence consisting of the subsequent words "is"…"is" is also classified to the JLC type Deeming Provision. However, this is not right. There is one NLC (natural language construct) missing, namely the word "deemed" (main term of the Deeming Provision JLC). To handle those wrong derivations we have to force that only one order of NLC's within each of the JLC's is applicable during the classification step. This idea is visualized in the under mentioned table.

Lexicon											
Id	**sem**	**cat**	**subcat**	**root**	**Gen**	**Case**	**Mood**	**Tense**	**Per**	**Num**	**Features**
5	**is**	V	AUX	**be**			INDICATIVE	PRESENT	3	S	JLCtype = DP_1
6	**deemed**	V	MAIN	**deem**			INDICATIVE	IMPERFECT	2	P	JLCtype = DP_2

Table 10. *Ordering the subsequent NLC's of the JLC type Deeming Provision*

Table 10 shows us the relevant NLC's for the JLC type Deeming Provision with within the feature column the attribute-value-pair extended with a subscript number to force a strict order of NLC's. Later, by the implementation of a Unification Algorithm we have to take care of these NLC order and existence. When one NLC is

missing or when the order of NLC's is different from the order of NLC's stated in our global model of JLC's, there is no relevant classification possible.

For example, when we have recognized the following main terms in a legal sentence

"by"	JLCtype={$Def2_1$}
"is"	JLCtype={$Def1_1$, $Def2_2$,DP_1}
"understood"	JLCtype={$Def2_3$}

we can classify this legal sentence to a Definition 2, because we can unify these three sets of JLC types by taking the intersection (and using the predefined order of the NLC's). Within the JLC type information of the word "is" there is stated that this word belongs to the JLC type Deeming Provision (DP_1), but the word "deemed" (DP_2) is missing, so no classification to this JLC type can be done.

11.1.2 Advantages/Disadvantages

The main advantage of this approach is that we can store all the JLC type information directly within the lexicon database table. The JLC information becomes easy administrable, extendable and transparent. When during further research (about the categorization of legal sentences) more JLC types can be defined, we do not have to add new production rules (rule based approach) to our translation engine, but we can add this new information directly to the lexicon database table. Another advantage is the fact that the parsing process becomes more efficient. The complete classification process is done by the recognition of only the main terms of the complete JLC structure. With this advantage, directly the main disadvantage comes across.

Because we only parse the main terms of each of the JLC's we do not have the extra information necessary to be able to generate complete formal models. Because our aim was to generate formal models from legal sentences, we also need the other JLC dependent language constructs to be able to construct the formal attributes, relations and associations. In addition, when we look at the structure of the JLC Definition 1 (`<subject> <is> <definition>`) the main term ("is") of the structure is not enough to generalize to one specific JLC. It can also classify to many other types.

11.2 Splitting of Juridical Information

A more elegant way of storing the JLC type information is to store it in a separate database. So we get two database tables connected by *primary*- and *foreign* keys (see Data Modelling: Primary and Foreign Keys 2004 [25]). One database table with lexical- and one with juridical information (see Figure 24).

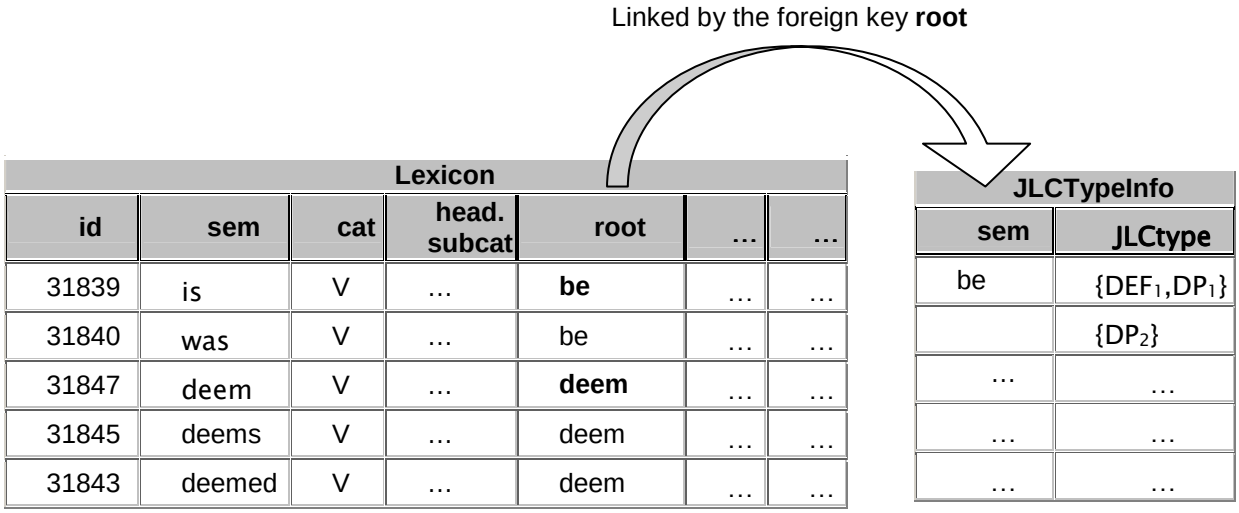


Figure 24. *The storage of juridical information in a separate database table*

Figure 24 shows us how the separation can be done. For every main term of the different JLC's, the main verb (root) of that verb is stored in a separate database. So for every grammatical form of the main verb we can refer to the root verb stored in the juridical database. During the parsing process, we can refer to the lexicon database for extra lexical information used for unification (for example if a verb is a link verb or a transitive verb etcetera).

The sem-feature is set as the primary key of the *JLCTypeInfo*-table, because this value is unique. The foreign key of the lexicon table is the root-feature. The id-feature of the lexicon table is the primary key.

By splitting up both database tables and connecting them by using the relevant keys we get a more transparent and efficient representation of lexical and juridical information.

11.3 Best of two worlds

Maybe in the future we can make use of the dynamic storage possibility within the lexicon database to specify new production rules (combination of both approaches). In the current implementation of the ePOWER Workbench there is functionality to use the JLC type information directly within the production rules as dynamic information (see Figure 25). We can add more unification rules to the relevant production rules of a specific JLC to add the dynamic JLC information of the main terms. Figure 25 visualizes this idea.

S -> NP_1 **V_2** (XLIST_3) **V_4** XLIST_5

<S inResult> = true
<S s_order> = sv
<V_2 features JLCtype> = DP_1
<V_2 head subcat> = AUX
<V_4 features JLCtype> = DP_2
<V_4 head subcat> = MAIN
<S head agr> = <NP_1 head agr>
<S head agr> = <V_2 head agr>
<S sem subject> = <NP_1 sem>
<S sem dp_part1> = <V_2 sem>
<S sem time_period> = <XLIST_3 sem>
<S sem dp_part2> = <V_4 sem>
<S sem fiction> = <XLIST_5 sem>

Figure 25. *The production rule of the JLC type Deeming Provision with dynamic JLC information*

The above figure shows us the adapted production rule of the Deeming Provision where, for the main terms, special unification rules are made. The information for these unification rules directly is derived from the lexicon database table. So, the main term JLC information becomes variable (dynamic). When for example by future research the Deeming Provision structure is adapted (so there are more words possible for each of the main terms), this change can easily be made within the lexicon database table.

For now, the functionality described in the second (non-rule based) approach can be seen as potential functionality for future development, when more information about the categorization of legal sentences is available.

12 Automated Rule Management: Grammar Editor

As mentioned before the manageability of the database of Grammar Rules is not efficient. When we want to add some new functionality to the ePOWER Workbench (the addition of new Production rules) we have to access the Production rule database table to make the necessary changes within the XML-script. Because working in a database, and creating complete new XML scripts is inefficient, the idea arose to automate this process. When a tool can administer the complete process of rule management it becomes more efficient and well organized, and we can add some checks to validate the input.

Because the initial ePOWER Workbench is implemented in .NET (see Microsoft .NET 2003 [23]) it seems a logical step to implement this component as a separate component within the ePOWER Workbench (see Appendix E for the complete programming code).

In the next section, the architecture of the tool will be discussed.

12.1 Implementing the tool

In the first place, examination of the general structure of the Grammar- and Unification rules was necessary (the XML-script in the Production rule database table). During this examination step I found that there were some fixed elements within the global structure of the rules (Grammar- and Unification rules), which are always present when a new rule is added or modified. These are overhead due to the use of XML.

LHS	RHS
<Lhs> **<FeatureSet>** **<Feature name="inResult">** **<AtomicValue>**true | false**</AtomicValue>** **</Feature>** **<Feature name="cat">** **<AtomicValue>**name of the LHS**</AtomicValue>** **</Feature>** **<Feature name="sem">** **<ComplexValue>** (**<Feature name**="feature-name" **equationId=**"1"**>** (<ComplexValue></Complexvalue>) ? **</Feature>**)+ **</ComplexValue>** **</Feature>** **</FeatureSet>** **</Lhs>**	**<Rhs>** (**<RhsElement nothingAllowed=**"true | false">** (**<FeatureSet>** **<Feature name="cat">** **<AtomicValue>**NLC-name**</AtomicValue>** **</Feature>** (**<Feature name**="feature-name" **equationId=**"id">** (<ComplexValue></ComplexValue>)? **</Feature>**)+ **</FeatureSet>**)+ **</RhsElement>**)+ **</Rhs>**

Table 7. *The general structure of the LHS and RHS of each Production rule*

Table 7 shows us the general structure of the LHS (left-hand-side) and the RHS (right-hand-side) of each production rule. One can see that each LHS consists of a fixed *<Lhs></Lhs>* construct, which consists of a fixed *<FeatureSet></FeatureSet>* construct, which consists of two fixed *<Feature><AtomicValue>true|false</AtomicValue></Feature>* constructs (the first

one is the *inResult*-feature (visible or not visible in the final derivation tree), and the second one is the *cat*-feature (the name of the LHS)) and one fixed *<feature><ComplexValue></ComplexValue></Feature>* construct (the *sem*-feature), whereby the *<ComplexValue></ComplexValue>* consists of one or more *<feature><ComplexValue></ComplexValue></Feature>* constructs, whereby the *<ComplexValue></ComplexValue>* construct is optional. The RHS consists of a fixed *<Rhs></Rhs>* construct, which consists of one or more *<RhsElement></RhsElement>* constructs (this construct contains a *nothingAllowed*-feature to indicate if this RhsElement is optional or not), which consists of one or more *<FeatureSet></FeatureSet>* constructs, which consists of a fixed *<Feature><AtomicValue></AtomicValue></Feature>* construct (the cat-feature, the name of the NLC) and one or more *<Feature><ComplexValue></ComplexValue></Feature>* constructs, whereby the *<ComplexValue></ComplexValue>* constructs is optional.

All these fixed structures can be extracted when a tool administers the rule set, so when Production rules have to be modified the tool will generate the fixed elements (no overhead). The user only has to fill in the other (non-fixed or variable) elements.

After the extraction of the fixed structures the next step was finding some kind of data structure for the storage of all the elements where all subsequent rules are built upon (A Grammar rule consists of one or more Grammar elements[23], each of which consisting of one or more Grammar features[24]) within the Production rule database table.

The general way to do this is the usage of a class model (see Appendix E, *GrammarClassModel.cs*). With a class model we can create a hierarchy within the different structures of the production rules. For our model we have to create four different classes within the GrammarClassmodel file: *GrammarRuleCollection* (see Appendix E, line number 29/206), *GrammarRule* (see Appendix E, line number 211/266), *GrammarElement* (see Appendix E, line number 369/441) and *GrammarFeature* (see Appendix E, line number 444/560).

The first class model is the main part of the class model. This part holds the complete Production rule collection in a .NET ArrayList [24] (see Appendix E, line number 61). Also, functionality of adding/sorting new rules (see Appendix E, line number 72/83) can be found in this project file. When new rules are added or rules are modified within the model, these modifications have to be saved in the class model. In the class model we can found the method *save()* (see Appendix E, line number 88/154) which can be used to assimilate all the modifications by writing all modified rules to the relevant database table (in other words the generation of XML script[25]). Other functionality within this class is considered as irrelevant at this moment (like validation and other event handling).

[23] A Grammar element can be used as being a LHS (a single element with a name and additional Grammar features) or a RHS (a list of Grammar elements each with a name and additional Grammar features). A synonym for Grammar elements is NLC (Natural Language Constructs, see Section 6.2).

[24] A synonym for Grammar features is Unification rules (see Section 6.2). They are used in the same way.

[25] The generation of the XML script is delegated to all the subelements (bottom-up). The main class in the class model collects the XML script from each of the Grammar Rules, the Grammar Rules generate the XML

The next three class model parts are the classes with the functionality of the specific rules. Each of these classes consists of three fixed parts:

- An ArrayList with the necessary subelements
- The method for the generation of the XML script
- A HTML generator (Pretty Printer, will be explained in the next chapter)

For the first class in the class model, GrammarRule, the ArrayList can be found in Appendix E, line number 216 (for the creation of the right-hand-side). The method for the XML generation can be found in Appendix E, line number 234/277 and 306/330. For the second class, GrammarElement, the ArrayList can be found in Appendix E, line number 374 (for the creation of a set of features). The method for the XML generation can be found in Appendix E, line number 432/441. For the last class, GrammarFeature the ArrayList is stated in Appendix E, line number 451. The XML generation method can be found in Appendix E, line number 528/560.

After we have created a way to store all the different Production rules in a class model the next step for the creation of the Rule Management Tool is finding a way to visualize the content of the Production rule set (in other words the User Interface).

After an examination of the different rules I have decided to visualize the complete rule set in some tree-like structure. The advantage of a tree structure is the well-organized, collapsible way to store complex data. Another argument to use a tree like structure is the presence of nesting within the rule set (as mentioned before). The *GrammarTreeModel* class (see Appendix E) contains the functionality to generate the tree structure from all the rules stored in the class model. Figure 26 shows us a screenshot of the final tree view of the Production rules.

Other classes used are *GrammarElementEdit*.cs (trapping the mouse clicks (right-clicks) on each GrammarElement), *GrammarFeatureEdit*.cs (trapping the mouse clicks (right-clicks) on each GrammarFeature) and *GrammarRuleEdit*.cs (trapping the mouse clicks (right-clicks) on each GrammarRule). The programming code for the main form is described in class *GrammarForm*.cs. The programming code for all these classes can be found in Appendix E.

script by collecting it from the Grammar Element class and the Grammar Element class generates the XML script from the Grammar Feature class.

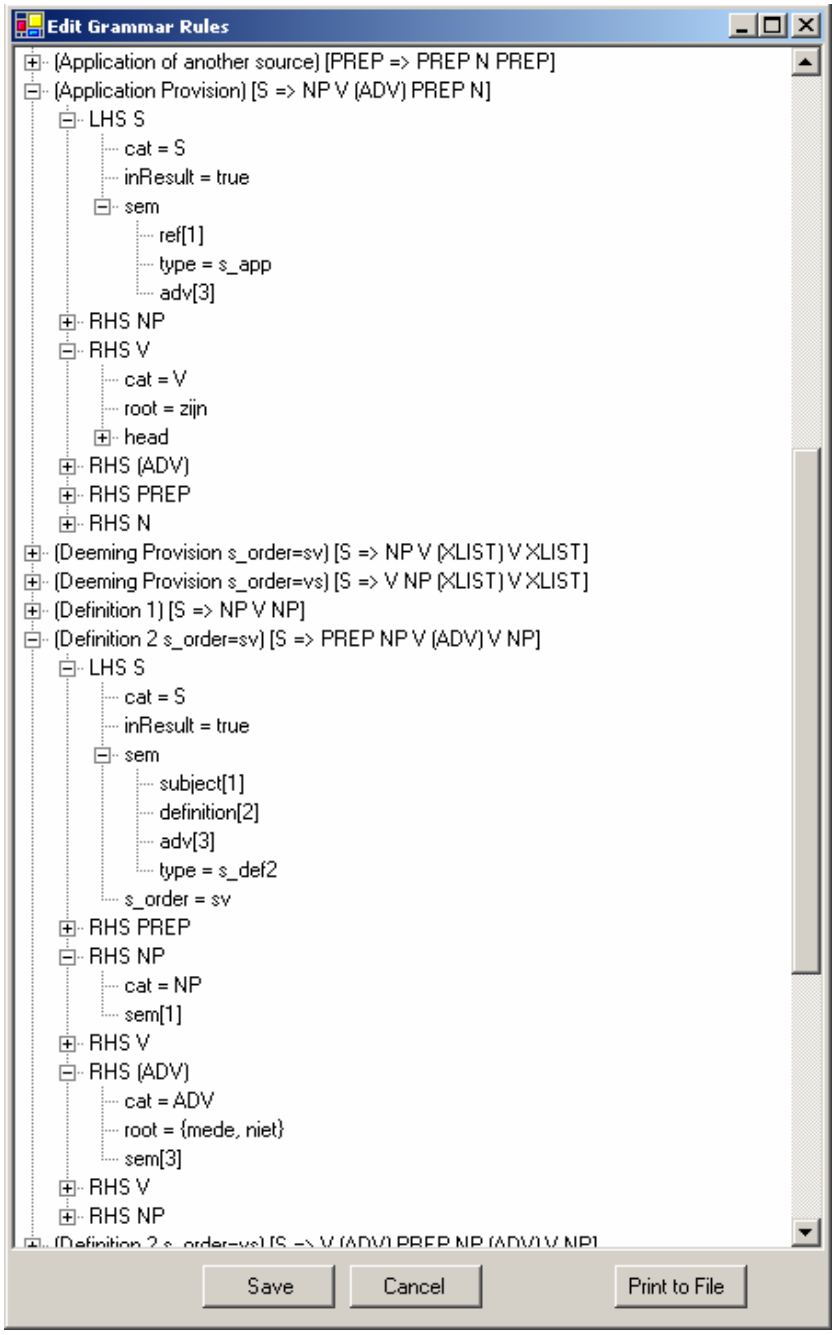


Figure 26. *A screenshot of the ePOWER Workbench Grammar Editor tool to clarify the usage of a tree structure to visualize the rule set.*

12.2 Grammar Editor Screenshots

The combination of all the mentioned classes has resulted in a Grammar Editor tool, which can be used to add, delete and edit functionality within the ePOWER Workbench. The functionality of the User Interface of the Grammar Editor tool is clarified by showing some subsequent screenshots (see Appendix D).

In the next chapter, I will clarify the implementation and usage of the Pretty Printer.

13 Pretty Printer

In Chapter 12, we discussed a tool to manage the complete Production rule set in an efficient way. In this tool, a *pretty printer* is included for the purpose of presenting a global view of the implemented Production rule set.

One of the advantages of such a pretty printer is the fact that during the implementation process (in particular during the generation of the Translation patterns) the possibility arises that you can easily refer to the global view of the relevant production rule information to track down the information, which has to be translated[26].

13.1 Implementing the Pretty Printer

In the previous chapter, a detailed description is given of the complete Grammar Editor .NET component within the ePOWER Workbench. In that chapter there is mentioned that each of the class model classes consists of three important methods. The first two are discussed, but the last one (the generation of the HTML of each of the class objects) isn't discussed already. This last part will be discussed in this section.

The Pretty Printer, also known as the HTML generator, is built by adding extra functionality to each of the classes of the class model (GrammarClassModel.cs, see Appendix E). This complete process is delegated (bottom-up process) from the root of the class model, the GrammarRuleCollection class. This class contains the method print (see Appendix E, line number 158/206), which collects all the information from his collection (ArrayList of GrammarRules) and prints it to a html-file (see Appendix E, line number 190/194). Because this process is delegated, also his child nodes have this print method. For the print method of each of the other classes in the class model see Appendix E (GrammarRule, line number 282/306; GrammarElement, line number 427/447; GrammarFeature, line number 507/532).

When the print method of the GrammarRuleCollection class is invoked all information is collected and subsequently printed to a html-file (see Appendix E, line number 197/204). The content of the html-file will also be displayed to the user.

[26] Because the Translation Patterns are built upon the information parsed by the Production rules there is a dependency between those two language dependent parts. From this view possessing a global view of the production rules seems useful.

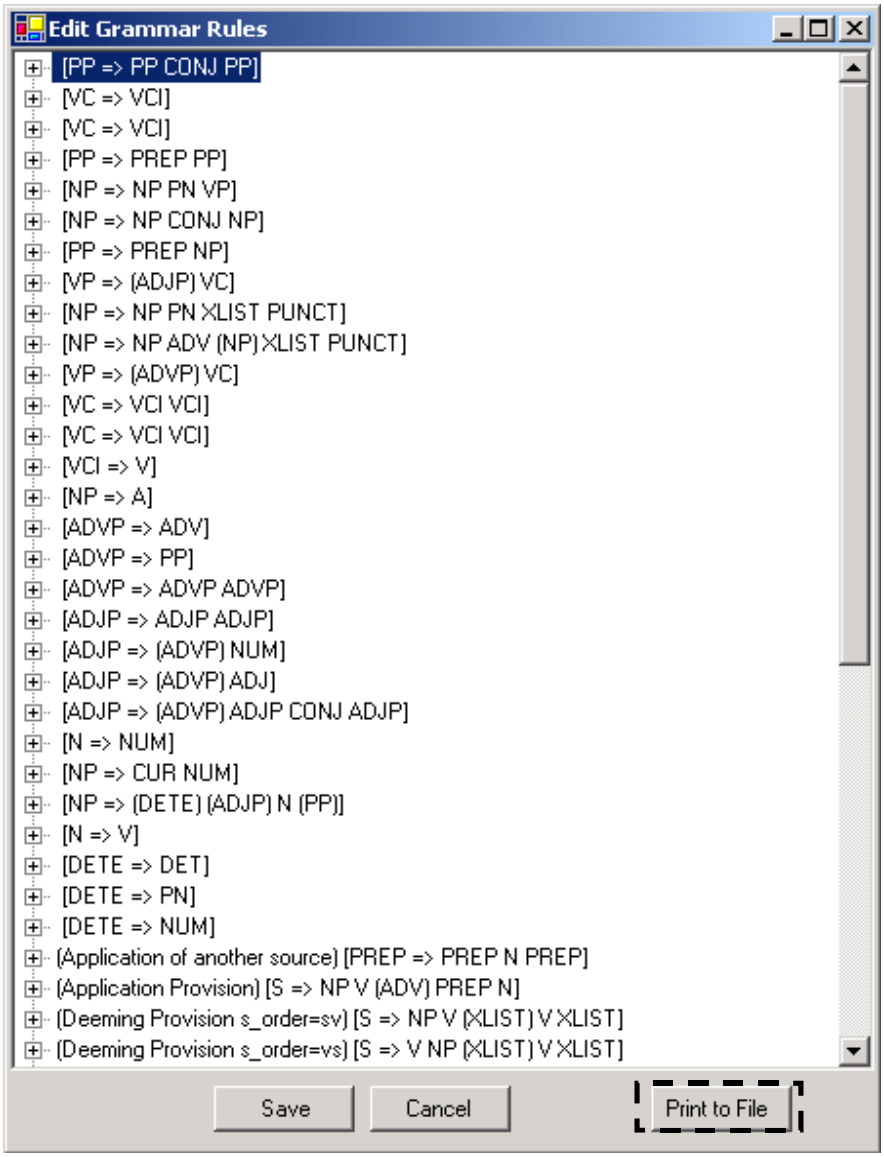


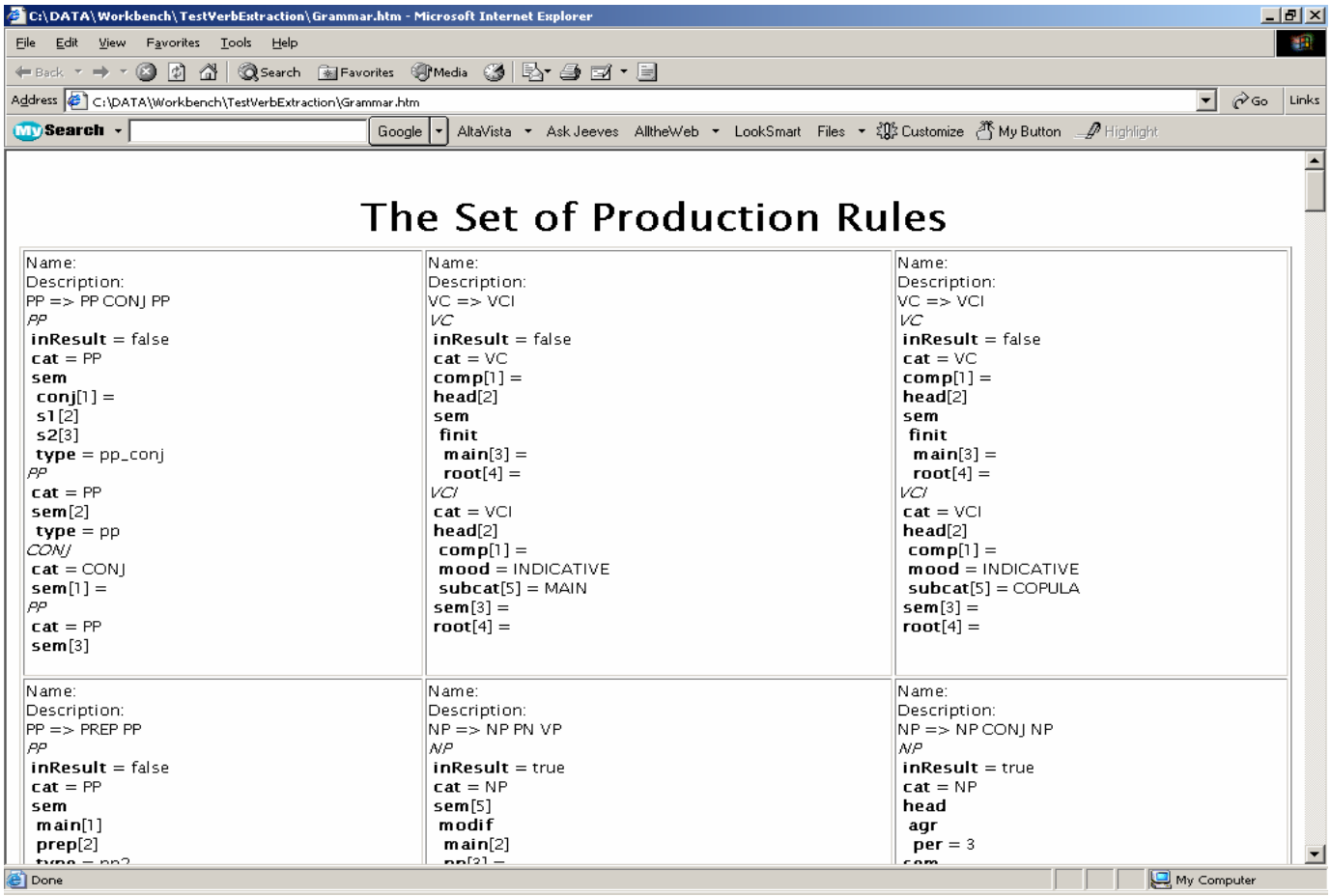
Figure 27. *A screenshot of the Rule Management tool with, for the execution of the Pretty Printer, an extra button ("Print to File").*

In the global User Interface (GrammarForm.cs, see Appendix E) an extra button is added to execute the Pretty Printer (see foregoing Figure 27).

In the next section, a screenshot of the output of the Pretty Printer is shown.

13.2 *Screenshot of the Output*

The screenshot below shows us the output of the Pretty Printer.



The screenshot shows us the content of the Grammar.html file, which has been built during the execution of the Pretty Print application.

This output has become very useful during the implementation of the translation patterns. There is a clear overview of all the stored information (the information parsed by application of the production rules), so the Pretty Printer has resulted in a more efficient and transparent way of implementing the translation patterns.

14 Related Work

By taking a closer look at alternative specifications and tools used in other projects, we can get an idea about the other possible solution methods of this specific problem. Maybe in the future, some of these alternative specifications can be used in the ePOWER Workbench translation engine.

Often, parsers are categorized according to the sets of languages that they can parse. The Chomsky hierarchy (see The Free Dictionary.com 2004 [38]) distinguishes between four types of families of languages: the regular languages, the context-free languages, the context-sensitive languages and the recursive enumerable languages (see Figure 28).

Grammar	Languages	Automaton	Production rules
Type-0	Recursively enumerable	Turing machine	No restrictions
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow aB$ $A \rightarrow a$

Figure 28. *The Chomsky Hierarchy (see The Free Dictionary.com website 2004 [38]).*

Our production rules also form a context-free grammar. However, the addition of the different unification rules gives us additional flexibility and helps us cope with, e.g. ambiguity. Unification in this context is similar to the use of semantic conditions in attribute grammars (see Knuth, D., E., 1968. [35]). We are not the first to use a formalism based on context-free grammars and attribute grammars for natural language processing (see e.g. The AGFL-project [36]). These formalisms are also prominent in the area of compiler construction.

The use of a lexicon is an important aspect of the Categorial Grammars (see Houtman, J. 1994 [39] and Pearson, J. 2003 [40]). The use of a lexicon combined with a fixed set of deduction rules makes categorial grammars easy to extend. Within this technique, there are no production rules, such as they exist in our tool.

A totally different viewpoint can be found in the paper of Costa F., Frasconi, P., Lombardo, V., Soda, G. 2001 [41]. This paper describes the development of novel algorithmic ideas for building a natural language parser by using recursive neural networks, grounded upon the hypothesis of incrementality.

Hellwig, P. 2002 [42] discusses many implementation and algorithms for context-free grammars (also, in combination with unification), and could be studied in the future to improve the current implementation.

15 Conclusion

During this thesis research, I have extended the ePOWER Workbench with the functionality necessary for the second step towards automated norm extraction from legal texts. This is called verb-phrase extraction. The ePOWER Workbench, as it is at this moment, can be used to recognize and translate a subset of all possible legal sentences described in the Dutch legislation into a formal model[27].

So, during this thesis research I have given evidence that supported both hypotheses stated in Section 3.1. My hypothesis for the recognition step of the verb-phrase extraction was as follows:

> *When examining the (limited) set of predefined natural language constructs (JLC's) defined by Emiel de Maat, special parse rules can be generated to extract the necessary legal knowledge from the legal sentences.*

My hypothesis for the translation step of the verb-phrase extraction was:

> *After the application of the parse rules, special translation patterns can be applied to generate the relevant formal models (expressed in UML/OCL).*

In Chapters 5, 6, and 7, I have discussed how the (limited) set of predefined natural language constructs (see Appendix A for the complete set of global structures defined for each of the different JLC's), defined by De Maat 2003 [6], can be used to generate the production rules for the recognition of all the normative elements from the legal texts (see Appendix C for the final set of production rules made for each of the different JLC's). In these chapters, it was described how to generate the final formal models by implementing the relevant translation patterns.

Therefore we can conclude that the legal sentences, although they are expressed in natural language, provide us with enough syntactical clues (found by De Maat 2003 [6]) to identify normative elements and consequently provide us with the handles to build an automated norm extraction tool.

As mentioned at the beginning of this thesis, the main target of the (E-)POWER program was to generate an environment supporting the generation of knowledge components (from normative knowledge sources represented in document form, via a formal model to a knowledge-based component (i.e. a piece of software able to make inferences about a certain regulatory domain)).

The ePOWER Workbench can therefore be seen as a starting point for the implementation of normative reasoning applications (applications that have the ability to reason about cases). It generates formal models from the normative knowledge sources (legal sentences).

[27] This is based on the current set of JLC's (Juridical (Natural) Language Constructs), but we have no reason to believe that different constructions would not fit in our framework.

The next step for the generation of knowledge components from the formal models (generated by the automated norm extraction tool), is the generation of programming code from a well typed (a type checker) OCL expression (see the thesis of Faridah Liduan 2004 [31], who has implemented such a code generator and type checker). At this moment, the code generator accepts OCL expressions and generates an intermediate language RBML (a rule-based XML document). This intermediate language can, by further research, be used to generate the necessary code for the implementation of a knowledge application. This will complete the (E-)POWER approach. Of course, improvements on each subsequent step are necessary to be able to fully rely on each of the different intermediate results.

At the end of my thesis research, the automated norm extraction tool has all the functionality necessary to recognize and translate legal sentences to a formal model. By introduction of this tool, a couple of advantages arise: the tool helps to reduce modelling time and effort while inter-coder dependencies diminish. When the formal models are made by hand (by experts), there is no guarantee that the generated formal models are similar. Afterwards, we have to check if the formal models are correct. When introducing an automated tool we generate the formal models consistently.

The final norm extraction tool is still in an early stage of development and still has to prove its benefit. I am however convinced that although I do not claim 100% recognition, a significant reduction of knowledge analysis effort (and further improvements in reducing inter-coder independencies) is achievable. Besides the advantage that this tool helps us to reduce modelling time and effort, there is also the advantage of reducing maintenance costs and total cost of ownership of the IT-service build upon the models produced this way. But in the beginning the aim of the (E-)POWER program was to generate a first version of a tool that supports automated norm extraction, so my approach can be seen as a successful one.

16 Recommendations for Further Research

In this chapter, some recommendations will be discussed for future development. During this thesis research, I have tackled the second problem of automated norm extraction, namely verb-phrase extraction. The ePOWER Workbench as it is today has all the functionality to recognize and translate a subset of all possible legal sentences as they occur in the Dutch legislation. Future development will result in more knowledge about normative reasoning with legal sentences and thus in a more useful tool for automated norm extraction.

16.1 Progressive Deconstructing of Abstract Language Constructs

During preliminary research done by De Maat [6], a couple of legal sentences have been chosen from a subset of the complete legislation (the law on income taxes from 2001, IB2001). By examining this subset, De Maat has categorized these legal sentences into a (limited) set of predefined natural language constructs (also known as JLC's, see Chapter 5) which can be used to define the legal sentences[28] (i.e. legal norms).

My thesis research, extending the initial ePOWER Workbench with functionality for verb-phrase extraction, is based upon these different categorizations[29]. So, at this moment the ePOWER Workbench is limited to recognize and translate a subset of all possible legal sentences. In addition, when we look at the categorizations built by De Maat some JLC's are specified in a very general, concise way. For example, when we look at the global structure and the production rule of the JLC Deeming Provision (see Figure 29 and 30) we can see that for the language construct other than the main terms only the production rule X_LIST (see Chapter 9) is used for extraction.

```
<subject>[wordt]<denotation of time period>[geacht]<fiction>
```

Figure 29. *The global structure of the JLC Deeming Provision with the main terms marked bold.*

When some legal sentence is recognized as being of JLC type Deeming Provision the specific intermediate language constructs are recognized by application of the X_LIST rule and subsequently concatenated for the generation of formal attributes. One can understand that there is always a possibility that, within these language constructs more relevant formal elements can be found (like classes, attributes, relations, associations etcetera). At this moment, no further specification of the sub constructs is available. Therefore, during my implementation, no further categorization of the abstract language constructs (within the legal sentence) is

[28] De Maat has examined a subset of all possible legal sentences occurring in the chosen law type. By this fact, there is not enough legal knowledge to recognize and translate all possible legal sentences occurring in the Dutch legislation. When new knowledge becomes available this can easily be added to the ePOWER Workbench translation engine by adding new production rules and translation patterns.

[29] Some categorizations (JLC's) are adapted and some categorizations are bundled to one single JLC. These adaptations are made because of implementation reasons. For the final set of generated production rules and translation patterns, see Appendix C and F.

applied. At this point, not enough knowledge is available to handle the subordinate clauses. They are recognized in the current tool, but their relation to the main terms (specified in the main sentence JLC's) is not determined.

S -> NP_1 V_2 **(XLIST_3)** V_4 **XLIST**_5	
<S inResult>	= true
<S sem type>	= s_dp
<S s_order>	= sv
<V_2 root>	= worden
<V_2 head subcat>	= AUX
<V_4 root>	= achten
<V_4 head subcat>	= MAIN
<S head agr>	= <NP_1 head agr>
<S head agr>	= <V_2 head agr>
<S sem subject>	= <NP_1 sem>
<S sem dp_part1>	= <V_2 sem>
<S sem time_period>	= **<XLIST_3** sem>
<S sem dp_part2>	= <V_4 sem>
<S sem fiction>	= <**XLIST_5** sem>

Figure 30. The production rule for the JLC Deeming Provision (s_order=sv), with for the recognition of the "denotation of time period" and "fiction" constructs the NLC's marked bold.

Later, when more knowledge about the subsequent language constructs of each of the JLC's becomes available, we can add new production rules (or adapt the relevant ones) and translation patterns to the ePOWER Workbench. In Chapter 7 a detailed description is given about how new knowledge can be added to the ePOWER Workbench translation engine by examining the global structure of each of the different JLC's. In addition, by using the Grammar Editor (or Automated Rule Management tool, see Chapter 12) new available knowledge can easily be entered into the ePOWER Workbench translation engine.

At this moment, I have introduced the X_LIST production rule to be able to recognize complete legal sentences. When we leave out this rule the translation engine is not able to apply the different production rules for each of the JLC's, because there is no information about the deeper structure of each of the intermediate language constructs. So, the X_LIST production rule is introduced to finish a first tool for automated norm extraction. The tool implemented during my thesis research can be used as starting point for further development within this field of science.

16.2 Automated Pattern Management

Another recommendation for future development is the generation of an editing tool to efficiently administer the set of translation patterns. At this moment, the translation patterns are specified by Visual Basic scripts, stored in a separate database (the pattern table in the translate-nl database, see Section 6.3).

Because these scripts are stored in a separate database table, there are a couple of shortcomings.

The first problem of storing programming code in a database table is the fact that administering these scripts is very hard (a limited set of editing tools is available).

The second more important shortcoming of using a separate database for editing the translation patterns is the fact that possible errors (syntactic- or semantic errors) are not recognized at the end of each adjustment. Possible errors within the translation scripts are only recognized by the direct application of them.

During the translation step (generating the formal models), the relevant scripts are collected from the database and directly applied. At this moment the ePOWER Workbench doesn't have any functionality for pre-processing the scripts. When there are some errors made during the generation of the translation pattern, the ePOWER Workbench simply crashes, so no detailed error messages appear to the user (no feedback). This is very inefficient because you simply do not have the exact position from where the exception is thrown. A more practical editing tool seems necessary.

A possible solution can be found in implementing an editing tool as a separate component of the ePOWER Workbench application. Like the tool developed to manage the set of production rules (discussed in Chapter 12) we can also implement such a tool in C# which can be used to edit the translation patterns in a more transparent way. Implementing this tool in .NET (the development environment of the ePOWER Workbench application) also has the advantage that we can add Visual Basic .NET functionality to the editor. One can think of syntax highlighting, auto-completion and macros. In addition, functionality can be added for pre-processing the edited scripts before they become available in the ePOWER Workbench translation engine. With this, better error messages can be returned. The user can detect and handle his errors in a more reliable and transparent environment than what is available in the current ePOWER Workbench norm extraction tool.

16.3 Transitive and Intransitive Verbs

Another recommendation for future development is the introduction of transitive- and intransitive verbs. Transitive verbs are verbs with some kind of special property. In the first place, a transitive verb is an action verb. Secondly, it requires a direct object to complete its meaning in the sentence. In other words, the action of the verb is transferred to the object directly (see the Transitive Verbs 2000 website [26]). In the following examples (taken from the website), the usage of the transitive verb is clarified. The transitive verb is marked bold and the direct object is underlined.

*The judge **sentences** the man to five years in prison.*

- The subject (the judge) applies an action (sentences) to a direct object (the man).

*The attorney **has revealed** the bad news.*

- The subject (the attorney) has transferred an action (revealed) to a direct object (bad news).

*The defendant **could** not **provide** an alibi.*

- The subject (the defendant) will transmit an action (could provide) to a direct object (an alibi).

The above examples can be used to visualize the global meaning of the usage of the properties of the transitive verbs.

Now let us look at the general usage of intransitive verbs. Intransitive verbs are the opposite of the transitive verbs. An intransitive verb is also an action verb, but it does *not* have a direct object. The action ends rather than being transferred to some person or object, or is modified by an adverb or adverb phrase.

To determine whether a verb is intransitive you have to ask whether the action is done in some way, in some direction or to some degree. Does anything receive the action of the verb? If it does, then the verb is transitive and the person or thing that receives its action is the direct object (see the Intransitive Verbs 2000 website [27]). In the following examples (taken from the website), the usage of intransitive verbs is clarified. The intransitive verb is marked bold and the modifier is underlined.

*The man **decided** against a plea bargain.*

- The subject (the man) did something (decided) a particular way (against).

*He **refused** because of his immaturity, not his lack of contrition.*

- The subject (He) did something (refused) for a particular reason (because of his immaturity).

*Alice **complained** bitterly.*

- The subject (Alice) did something (complained) to a particular degree (bitterly).

*At the end of the Roaring '20s, the incarceration index **rose** slightly.*

- The subject (the index) did something (rose) in a particular direction (slightly).

*When faced with the problem, the scholar **paused**.*

- The subject (scholar) did something (paused) at a particular time (when faced with the problem).

*Earl **fell**.*

- The subject (Earl) did something (fell) and the action did not transfer to someone or something.

The above examples can be used to visualize the global meaning of the usage of the properties of intransitive verbs.

In our case, these special properties can be used to extend the ePOWER Workbench norm extraction tool with the knowledge for the generation of formal associations. One can think of the generation of a production rule of the following form:

$$S \rightarrow NP_1 \ \mathbf{V} \ NP_2$$

The first noun-phrase is the *subject* of the sentence and the second noun-phrase is the *direct-object* of the legal sentence. The verb (V) is the language construct, which we can use to extract the information about transitive- and intransitive verbs (the existence of a transitive verb always forces a relation between the subject and the direct-object of the legal sentence and we can generate a association between them).

To be able to use this production rule some extra information has to be added to the relevant verbs. One can think of an extra feature, which we have to add to all the transitive verbs and intransitive verbs. This extra feature could be in the form of an attribute-value-pair *Transitive=true/false* added to the features column of the lexicon database table. During the application of the production rules, we now have the possibility to refer to the lexicon to check if the recognized verb has the transitive verb property. If so, we can generate a formal association from the extracted language constructs.

Something we have to keep in our mind is the recognition of the language construct "is". This word can be used as part of the global structure of the JLC Definition 1 (<subject>[**is**]<definition>, see Appendix A) or it can be used as part of the already mentioned global structure used for the recognition of the transitive and intransitive verbs. Special care seems necessary.

In addition, when applying all this knowledge, during the recognition step of the norm extraction tool, we have to check if this extra knowledge will lead to correct formal associations. More specific research on the usage of these grammatical properties seems necessary.

16.4 Fixed Verb-Preposition Couples

Another grammatical property that we can use for future development is the usage of fixed verb/preposition couples (see Verb and Preposition Collocations 2002 [28]). Table 11 shows us a subset of all the possible verb/preposition couples that commonly appear together in the English language.

accuse (someone) of ([doing] something)
add (something) to (something else)
admire (someone) for ([doing] something)
agree on (topic)
agree with (someone)
allow to ([do] something)
apologize to (someone) for ([doing] something)
apply to (a place) for (something)
approve of (something)
argue with (someone) about (topic)
arrive at (a building, room, site, event)
arrive in (a city, country)
ask (someone) about (someone/topic)
ask (someone) for (something)

believe in (something)
belong to (someone)
blame (someone) for ([doing] something)
borrow (something) from (someone)

care about (someone/something/topic)
comment on (topic)
compare (something) to/with (something else)
complain to (someone) about (something)
concentrate on ([doing] something)
congratulate (someone) for/on ([doing] something)
consist of (some things)
consent to ([doing] something)
contribute to (something)
count on (someone) to (do something)
cover (something) with (something else)

decide on (topic)
depend on (someone) for (something)
discuss (something) with (someone)
distinguish (something) from (something else)
dream about/of (someone/something)

escape from (somewhere)
explain (topic) to (someone)
excuse (someone) for ([doing] something)

forgive (someone for ([doing] something)

get rid of (something)
graduate from (a place)

happen to (someone)
help (someone) with (something)
hide (something) from (someone)

insist (up)on (something)
introduce (someone) to (someone else)
invite (someone) to (an event)

keep (something) for (someone)

matter to (someone)

object to (something)

participate in (something)
pay (price) for (something)
pray for (someone/something)
prefer (something) to (something else)
prevent (someone) from ([doing] something)
prohibit (someone) from ([doing] something)
protect (someone) from (something)
provide (someone) with (something)

recover from (something)
rely (up)on (someone/something)
remind (someone) of (something)
rescue (someone) from (something)
respond to (someone/something)

save (someone) from (something)
search for (something)
separate (something) from (something else)
scold (someone) for ([doing] something)
shoot (someone) with (something)
smile at (someone) for ([doing] something)
speak to/with (someone) about (topic) /br> stare at (something/someone)
stop (someone) from ([doing] something)
subscribe to (something)
substitute (something) for (something else/someone)
subtract (something) from (something else)
succeed in ([doing] something)
suffer from (something)

take advantage of (someone/something/ situation)
take care of (something/someone)
talk to/with (someone) about (topic)
thank (someone) for ([doing] something)
travel to (somewhere)

vote for (someone)
vouch for (someone)

wait for (someone/something)
wish for (something)
work for (company/something/someone)

Table 11. *A list of verbs and prepositions which commonly appear together in the English language, with the relevant couples for the relevant example marked bold (see website [28]).*

Initially we can add this knowledge to the ePOWER Workbench translation engine for the generation of formal associations. However, I think, this will become one of the main targets in getting around the ambiguity problem (or the PP-attachment problem). For example, when we want to translate the following sentence

it is not allowed to shoot a man with a gun.

this sentence has two possible interpretations. In the first place, one can interpret that it is not allowed to use a gun to shoot a man. The other can interpret that it is not allowed to shoot a man who has a gun. The difference in interpretation lies in the attachment of the preposition (PP-Attachment problem). Because this sentence will result in two different derivation trees (parse trees), we call this the ambiguity problem. In my opinion, the foregoing knowledge about the verb/preposition couples can help to get around ambiguity in some extent. For example, when we apply this knowledge to the foregoing example we can extract two verb/preposition couples, which we can use to reason about the exact meaning of the complete sentence. The first one is "allowed to" (allow to ([do] something)) and the second one is "shoot with" (shoot (someone) with (something)). To determine the exact meaning of the above sentence we can use the global meaning of the both verb/preposition couples. The first couple does not give us enough information to conclude one derivation, because the ambiguity lies in the second phrase (the "something"-part of the first couple) of the sentence ("to shoot a man with a gun"). When we look at the global structure of the second couple, there is stated that you have to shoot someone with something. This can help us to conclude one derivation in the sense that in the first place we search for the someone-part (so finding a noun or noun-phrase which holds information about the person who will be shot) and after that we will search for the something-part (the thing where the someone-part will be shot with). In our case, we can use the information to conclude (forcing) that it is not allowed to shoot a man when we make use of a gun. (Of course, in some cases both derivations make sense, but since we work in the context of the law we do not expect ambiguity) By looking at the verb/preposition couples, we can try to figure out what the exact meaning is of the specific sentence. I can imagine that in my initial solution there are some shortcomings, but future development has to prove that we can use more grammatical knowledge about the language constructs to interpret the input sentences in a more efficient and transparent way without having the problem of ambiguity.

16.5 Enumerations

In the current norm extraction tool there is not enough knowledge (specification in the form of a global structure containing enumeration statements) available about the extraction of legal sentences which hold information in an enumerated way.

For example, when we want to translate the following legal sentence

IB 2001 Art 2.1 lid 1
Taxpayers for the income tax are natural persons who:
a. live in the Netherlands (native taxpayers) or
b. do not live in the Netherlands but do earn Dutch income (foreign taxpayers).

the parsing process will result in the recognition of the noun-phrases, but the translation engine doesn't have enough information about the recognition of enumeration statements for the recognition of the complete legal sentence. When we look at the above legal sentence we initially can apply the JLC Definition 1 (`<subject>[is|are]<definition>`, see Appendix A) to recognize the sub sentence "*Taxpayers for the income tax **are** natural persons*". The other information is in the form of an enumeration. Both the statements say something about the definition part of the JLC Definition 1, namely "*natural persons*", in the sentence that it adds some additional information which will result in the fact that this legal sentence only is applicable (restricted) to natural persons which have the further described properties (described in the enumeration). To recognize this sentence we have to add some more knowledge about the global structure of enumeration statements within the current global structures of the JLC's. Future research, about the existence and translation of enumeration statements should yield a way of how we can handle those statements. This can be done by detecting all possible enumeration statements used in the source documentation.

16.6 The NLC Formula

At this moment, a concise production rule is generated for the recognition of the NLC formula (like in the JLC Assignments, Changes and Comparison, see Appendix A). The tool has functionality to recognize legal sentences containing the following formula constructs:

x increased by *y* *x* decreased by *y*	By using the production rule 14 (Appendix C "Production rules for Verb-phrase Extraction") and translation pattern *type="np_formula"* (Appendix F)
at most *x* at least *x*	By using the production rule 25 (Appendix C) and translation pattern *type = "np" and root in ("bedrag", "waarde", "hoogte") and pp.prep = "van"* (Appendix F)

One can think of more than only these formula constructs (like "sum of *x* and *y*", "*x* divided by *y*", etcetera), but at this moment it is only possible to recognize and translate the former constructs to their relevant formal model elements. When by future research more legal knowledge about the global structure of each of the possible formula statements becomes available more production rules and translation patterns can be made.

16.7 Multiplicity in Associations

At this moment, the ePOWER Workbench translation engine can translate legal sentences, containing formal associations, into their relevant formal model. However, there is more knowledge necessary to complete the recognition of associations. Information about the multiplicity of the association is necessary (see the UML specification [9]). The general UML notation for an association can be found in Figure 31.

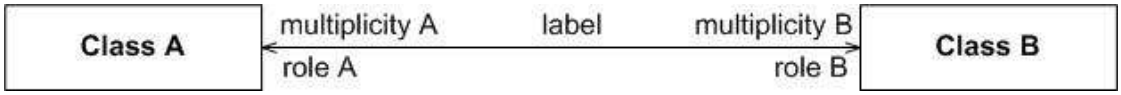


Figure 31. *General UML notation for associations.*

Figure 31 shows us that for the correct generation of the formal association between class A and B also information about the multiplicity of role A and role B has to be available. Table 12 depicts all possible multiplicity indicators. On both ends of the association, one of these indicators must be added.

Indicator	**Meaning**
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Table 12. *Multiplicity Indicators for associations.*

At this moment, associations are recognized consisting of two noun-phrases, which are related to each other in some kind of way. The information about the multiplicity of both the noun-phrases is not yet extracted from the legal sentence by the current translation engine. There is no global specification about how to extract multiplicity of associations from legal sentences yet.

My approach would be to examine the noun-phrases for the existence of some key words that indicate a multiplicity indicator. For example when the language constructs “many”, “a couple of”, “some”, “a set of”, etcetera can be extracted as being part of a noun-phrase this will in most cases result in the fact that the multiplicity indicator is of the form *. One can also think of language constructs, which can be used to recognize the other multiplicity indicators. However, at this

moment I think that the global meaning of the introduction of multiplicity of associations is clear.

16.8 Alternative Storage of the Lexical Data

One of the alternative solutions for warehousing and accessibility of all the words possible in the Dutch language (the lexicon) is the usage of the computational morphology technique[30] (see A Computational Morphology of English 2004 [29]). At this moment for each word and word form (like “is”, “was”, “were”, “been” and etcetera) a separate record in the database is made, because of the grammatical and lexical difference. By making use of the computational morphology technique only the singular form of a noun or verb is stored and in runtime the other forms can be computed. For example, we only have to store the verb-form “ren”[31] to calculate the verb-forms “rent (hij)” “rennen (wij)”, etcetera. We leave out the “en”-part of the infinitive part of all the different verbs occurring in the Dutch language to calculate all the other lexical forms of the verb.

The main advantage of such a technique is the fact that we need less storage space, but a bit more powerful CPU. At this moment, the ePOWER Workbench application does not make use of this technique. Maybe in the future we can add this functionality to the lexicon (of the Dutch language) for efficiency reasons, but at this moment this seems too much overhead.

16.9 Multiple Language Support

Finally, the ePOWER Workbench contains the information and functionality necessary to recognize and translate a subset of the Dutch legislation. The complete norm extraction tool as it is present in this version of the ePOWER Workbench is limited to the recognition of the Dutch language. Because the ePOWER Workbench is made for multiple language support it can easily be extended with the functionality necessary for the recognition and translation of other languages (see Chapter 6 for more detailed information about all relevant parts of the ePOWER Workbench).

At this moment the ePOWER Workbench uses only three language dependent parts namely, a lexicon (with all the possible words possible in the specific language), a set of Production rules (for the creation of the Grammar rules and Unification rules) and a set of Translation patterns (for the translation of the information parsed during the application of the Production rules). These three parts are necessary for the ePOWER Workbench to recognize a specific language. So, when these three language dependent parts are available for another language, the norm extraction tool can be used as well.

[30] The computational morphology technique is only applicable for the storage and accessibility of all the possible words occurring in the Dutch language. Therefore, in this section the examples are specified in Dutch. For the usage of the computational morphology technique for the English language I can refer to.
[31] The English main verb of the Dutch verb-form “ren” is “to run”.

16.10 Multiple Law Types Support

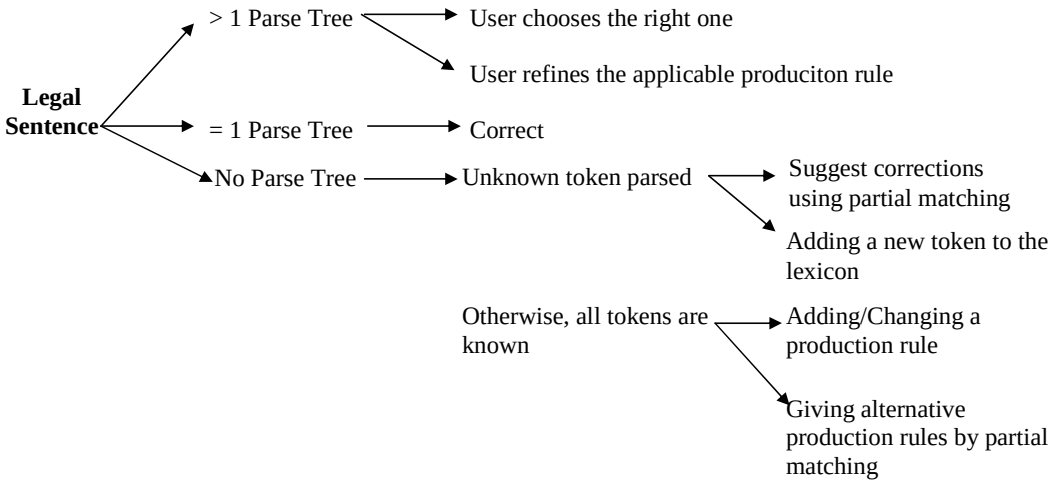
In the current version of the ePOWER Workbench only a subset of the possible legal sentences described in the law on income taxes from 2001 (Wet Inkomsten Belasting 2001, IB2001) can be recognized and translated. This because, my approach is based on the categorization of the different legal sentences (JLC's) defined by De Maat 2003 [6], occurring in the already mentioned law.

Maybe by future research the set of JLC's can be extended by determining more knowledge about the global structure of the legal sentences described in other law types.

Globally, this thesis research and aforegoing researches had made a step towards formalising legal knowledge using natural language processing.

16.11 Errors and other Classification Problems

The ePOWER Workbench only gives a result if a sentence can be classified to at least one JLC. If this is not possible, then at this moment no feedback is given to the user. Obviously, this situation needs improving. We suggest the following architecture for interaction with the user:



From this architecture, we can conclude that there are two types of errors. In the first place, there is a possibility that the legal sentence is incorrect. On the other hand, it is also possible that the current set of production rules and the lexicon are incomplete.

16.12 Improvements on our Implementation

In this section, we list a number of possible improvements of the current implementation of the ePOWER Workbench. Also, an alternative implementation technique, Attribute Grammars, will be discussed.

One of the shortcomings of the current implementation is the fact that there is more than one language[32] used in the translation tool. First, the parsing process is based on a set of Production rules. These rules are stored in a separate database (translate-*nl*, see Section 6.1) and described by the XML formalism (see Section 6.2).

When the parsing process is finished, the second step is to translate the derivation tree into a formal model. For this purpose, the translation tool makes use of the translation patterns, which are also stored in the same database. These translation patterns are in the form of a Visual Basic script (see Section 6.3). The rest of the functionality (see Figure 6) is implemented in the .NET environment (by the programming language C#). Therefore, we can conclude that there are three used, namely C#, XML and Visual Basic.

A disadvantage of this, is the fact that during the execution of the ePOWER Workbench tool, different languages have to be able to communicate with each other. For example, for the extraction of the normative legal constructs (the application of the production rules), the translation tool obtains the production rules from an XML file. Before these rules can be used the XML file should be parsed at runtime, so exceptions are possibly thrown during this process. No error messages appear when the user makes some adaptations to the production rule set. This is one of the main problems of the current version of the ePOWER Workbench. In Chapter 12 an editing tool is described, the Grammar Editor, which can be used to edit, delete and create production rules. The disadvantages of using the database for modification purposes are now dealt with. The problem of determining possible errors is still there, but at this moment the editing tool cannot compile XML script.

Almost the same problem arises during the application of the translation patterns. These patterns are described in Visual Basic and are stored in a database. Possible error messages only appear at runtime, and the user does not receive a clear error message with information about the place (in the code) from where the error originates. In addition, a lot of syntactical overhead is present when using Visual Basic script. For every production rule a translation pattern (stored in a different record in the database table) has been made. When some functionality (in the form of a programming method) is applicable for more than one translation pattern, we have to copy and paste this method in every record in the database where this functionality can be used. Reusing programming code is impossible, and we get duplication of effort and errors. Another disadvantage of using Visual Basic scripts is the fact that there is too much control (it is not as declarative as one could hope). In Section 16.2, a possible editing tool is described.

The main reason of the disadvantages mentioned above, is the fact that both these scripts are stored in a database. In general editing a record in a database is not efficient. There is no syntax highlighting, no auto-completion and no macros functionality. In addition, there is no support to determine errors (typing-, syntax- and type errors). At this moment, we have to make use of the debugging facility of the .NET environment to be able to determine the place where from a possible error is thrown.

[32] In this chapter, the word “languages” is used. In this case, also the word phrase “programming languages and formalisms” can be used.

After discussing the current implementation of the ePOWER Workbench, it is time to discuss a possible alternative implementation of the main problem. We simply want to make use of one single programming language (if necessary using a database) to describe the complete functionality of the ePOWER Workbench translation tool. For example, we really want to change the specification language used for the translation patterns into a more declarative language, like PROLOG, ML or HASKELL (most preferable a strongly typed language).

One of the possibilities is the usage of an Attibute Grammar for natural language processing (see Knuth, D., E., 1968 [35]). Attribute grammars are an extension of context-free grammars as a mechanism for the semantics of a context-free language within the syntax of the language (Mehdi Jazayeri, William F. Ogden, and William C. Rounds 1975 [34]). Usually, this language is more declarative than Visual Basic script and the C# language. In other words, an attribute grammar can be used to define semantic rules for a parse tree. In the Netherlands at the University of Nijmegen a project started called the AGFL-project (Affix Grammars over a Finite Lattice) which goal is the development of a technology for Natural Language Processing by using the Attribute Grammar technique (see The AGFL Grammar Work Lab 2004 [36]). This homepage gives an exact description about how this technique can be used for natural language purposes. A parser for the Dutch language is included as well as a parser for some other languages. Of course, we have to add some extra functionality to the parser to be able to translate the parsed information into a formal model. This can be done by specifying semantic rules (with an attribute grammar).

References

[1] Van Engers, T.M., Boekenoogen, M., 2003, Improving Legal Quality - an application report, in *Proceedings of ICAIL2003*, ISBN 1-58113-747-8, ACM Press.

[2] ePower Workbench 2.6, Belastingdienst Utrecht, Centrum voor Proces en Productontwikkeling (B/CPP), Utrecht, 2000

[3] USER DOCUMENTATION ePOWER Workbench 2.6, August 2003

[4] Organisatie van de Belastingdienst, 2004, website http://www.belastingdienst.nl/corpinfo/inhoud/inh_org_corp.html.

[5] E-POWER Homepage, 2004, E-POWER Consortium, website http://www.lri.jur.uva.nl/~epower/.

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

[7] Microsoft Office Word 2003 Support, Microsoft Corporation, 2004, website http://office.microsoft.com/assistance/topcategory.aspx?TopLevelCat=CH79001816&CTT=6&Origin=ES790020011043.

[8] Zwemmer, J.W., 2002, Belastingwetten 2002, SDU Uitgevers, Amersfoort, ISBN: 9076629781

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

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

[11] Van Engers, T.M., Glassée, Erwin, 2001, Facilitating the Legislation Process Using a Shared Conceptual Model, in *IEEE Intelligent Systems,* January/February 2001 p50-58.

[12] Egberts, Niels, 2004, Proces Modellering van Temporele Aspecten in Wet- en Regelgeving, Master Thesis, Universiteit van Utrecht, Department of Computer Science, Utrecht

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

[14] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991, *Object-Oriented Modelling and Design.* Englewood Cliffs NJ, Prentice-Hall.

[15] Frederiks, P., 1997, Object-oriented modeling based on information grammars. Nijmegen.

[16] Nijssen, G.M. 1989, Grondslagen van Bestuurlijke Informatiesystemen. Slenaken. Nijssen Adviesbureau voor Informatica.

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

[18] The Natural Language Processing Dictionary, *keyword "agreement",* Artificial Intelligence Group, School of Computer Science and Engineering, University

of NSW, Bill Wilson, 2003, website http://www.cse.unsw.edu.au/~billw/nlpdict.html.

[19] XML Developer Center, *The Language of Information Interchange*, 2004 Microsoft Corporation, website http://msdn.microsoft.com/xml/default.aspx

[20] Rule Based Systems, alison@ 1994, website *http://www.cee.hw.ac.uk/~alison/ai3notes/section2_4_4.html*.

[21] Coppen, P.A., Haeseryn, W., De Vriend, F., De determinator, *De Elektronische ANS (Algemene Nederlandse Spraakkunst)*, 2004, Stichting *ANS*, website http://oase.uci.kun.nl/~ans/e-ans/14/04/body.html.

[22] Visual Basic Language and Run-Time Reference, Nedcomp Hosting, website http://www.nedcomp.nl/support/origdocs/dotnetsdk/vblr7net/vboriVBLangRefTopNode.htm.

[23] .NET, 2003, Microsoft Corporation, website http://msdn.microsoft.com/.

[24] .NET Framework Class Library ArrayList class, 2004, Microsoft Corporation, website http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemcollectionsarraylistclasstopic.asp.

[25] Data Modelling: *Primary and Foreign keys*, Information Technology Services at The University of Texas at Austin, 2004, website http://www.utexas.edu/its/windows/database/datamodeling/dm/keys.html.

[26] Transitive Verbs, The Tongue Untied, *A guide to grammar, punctuation and style*, 2000, Kellee Weinhold, website http://grammar.uoregon.edu/verbs/transitive.html.

[27] Intransitive Verbs, The Tongue Untied, *A guide to grammar, punctuation and style*, 2000, Kellee Weinhold, website http://grammar.uoregon.edu/verbs/intransitive.html.

[28] Verb and Preposition Collocations, eslgold.com, 2002, website http://www.eslgold.com/site.jsp?resource=pag_stu_grammar_expl_exa_exer_hi_verb_prep.

[29] A Computational Morphology of English, 2004, SIL International, http://www.sil.org/pckimmo/v2/doc/englex.html.

[30] Unification-based syntactic parser PATR, SIL International Partners in Language Development, 2004, SIL International, website http://www.sil.org/computing/catalog/show_software.asp?id=37.

[31] Liduan, Faridah, 2004, Design and Implementation of a UML/OCL Compiler, Master Thesis, University of Utrecht, Department of Computer Science, Utrecht

[32] KDNet Symposium: "Knowledge-Based Services for the Public Sector", June 2004, http://symposium.kdnet.org/symposium/symposium.jsp.

[33] Van Engers, T.M., Sayah, K., Van Gog, R., De Maat, E., Automated Norm Extraction from Legal Texts, 2004, Proceedings KDNet Symposium

[34] Jazayeri, M., Ogden, W., M., Rounds, W., C., The intrinsically exponential complexity of the circularity problem for attribute grammars, *Communications of the Association for Computing Machinery*, 18(12):679-706, December 1975.

[35] Donald E. Knuth, D., E., Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127-145, 1968.

[36] The AGFL Grammar Work Lab, 2004, http://www.cs.kun.nl/agfl/

[37] Van Gog, R., Production rule Set and the Translation Patterns for the Noun-phrase Extraction, Unpublished.

[38] The Free Dictionary.com, *Chomsky Hierarchy*, 2004, Farlex, Inc., website http://encyclopedia.thefreedictionary.com/Chomsky%20hierarchy

[39] Houtman, J., Coordination and Constituency, *A Study in Categorial Grammar*, 1994, p. 19-51, University of Groningen

[40] Pearson, J., Natural Language Processing: Semantic Analysis, 2003, The College of New Jersey, Department of Computer Science

[41] Costa, F., Frasconi, P., Lombardo, V., Soda, G., Towards incremental parsing of natural language using recursive neural networks, 2001, University of Florence, Department of Systems and Computer Science

[42] Hellwig, P., Natural Language Parsers, *A Course in Cooking*, 2002, Heidelberg

Appendix A

In this thesis, I make use of the categorization of the different legal sentences (see Chapter 5) as it is described in the thesis research of Emiel de Maat [6]. This categorization specifies for each of the different legal sentence categories (also known as JLC's) a rule-based structure[33]. This rule-based structure is used to define Production rules to recognize and extract the necessary language constructs for the generation of the relevant formal model. In this appendix a description of the rule based structure of each of the different JLC types is depicted as it is described in the thesis of Emiel de Maat (with the main terms of the global structures marked bold).

Deeming Provision

<subject>**[wordt]**<denotation of time period>**[geacht]**<fiction>

Explicit Condition

[If]<subject><feature>
[Insofar]<subject><feature>

Implicit Condition (subordinate clause)

[who|which|that]<feature>

Definition

<subject>**[are|is]**<definition>	(1)
[By]<subject>**[is understood]**<definition>	(2)
[By]<subject>**[is also understood]**<definition>	(2 broaden)
[By]<subject>**[is not understood]**<definition>	(2 narrow)
[As]<term>**[is considered]**<new_term>	(3)
[As]<term>**[is also considered]**<new_term>	(3 broaden)
[As]<term>**[is not considered]**<new_term>	(3 narrow)
<new_term>**[is set to equal with]**<term>	(4)
<new_term>**[is qualified as]**<new_term>	(4)

[33] In the thesis of Emiel de Maat, a subset of all possible legal sentence categorizations is described. Further research on this subject is likely to lead to more knowledge about the global structure (categorization) of legal sentences. At this moment, I have used only the currently available knowledge about the global structure of the legal sentences.

Application Provision

<reference>**[applies]**

<reference>**[does not apply]**

Value Assignment, change and comparison

<subject>**[is|amounts to]**<formula>

<subject>**[is set to]**<formula>

Relations

[to apply (to)]

Scope Definitions

[For the application of]<reference><statement>

References

<term>**[as meant in]**<reference>

Application of another source

[due to application of]<reference>

[based on]<reference>

Appendix B

To be able to use the XLIST Grammar rule we have to add some extra functionality to the ePOWER Workbench. With this extra functionality every individual word, except words with the cat-feature is equal to PUNCT, has the X-type property. This will result in the fact that every word or a sequence of word can be parsed when no other specific Grammar rule is applicable (this becomes handy when a sequence of words has to be parsed for the generation of an attribute). In the programming code the adaptations can be found (marked bold).

Workbench.NaturalLanguage.Lexicon.Lexicon.*LookupLexemes*(StringCollection lexemesToLookup, string mode, CultureInfo culture)

```
public FeatureSetCollectionCollection LookupLexemes(StringCollection lexemesToLookup,string
mode,CultureInfo culture)
{
  FeatureSetCollectionCollection retVal = new FeatureSetCollectionCollection();
  if(lexemesToLookup.Count > 0)
  {
    DataTable LexiconTable = this.lexiconDAC.
       SelectLexiconEntriesForLexemes(lexemesToLookup,mode,culture).Tables[0];
    FeatureSetCollection fsc = null;

    foreach(string lexeme in lexemesToLookup)
    {
      if(retVal[lexeme] == null) //als nog niet opgezocht... (!)
      {
        string expr = "sem=" + "'" + this.lexiconDAC.EscapeSingleQuotes(lexeme,'\'') + "'";
        DataRow[] foundRows = LexiconTable.Select(expr);
        if(foundRows.Length!=0)
          fsc = GetFeatureSetsFromRows(foundRows);
        else
          fsc = ApplyLexiconAdditionRegExes(lexeme,mode,culture);
        if(fsc == null)
        {
         if(!this.htLexiconSupplements.ContainsKey(mode+"-"+culture.Name))
          this.LoadLexiconSupplements(mode,culture);
        ArrayList lexiconSupplements = (ArrayList)this.htLexiconSupplements[mode+"-
        "+culture.Name];
        foreach(ILexiconSupplement lexiconSupplement in lexiconSupplements)
        {
          fsc = lexiconSupplement.ProcessLexeme(lexeme, mode, culture);
           if(fsc!=null) break;
        }
       }
       if(fsc == null)
         fsc = new FeatureSetCollection();
       foreach(FeatureCollection fc in fsc)
       {
           if(this.lexemeFeatureName != null)
            fc.Add(lexemeFeatureName, new FeatureValue(lexeme));
          ApplyIncrementalRegExes(lexeme, fc, mode, culture);
       }
       // START X Category
       // Deze code zorgt dat elk woord altijd ook als categorie X wordt toegevoegd.
       if (".,;()?!".IndexOf(lexeme)<0)
       {
         FeatureCollection fcX = new FeatureCollection();
         fcX.Add("sem", new FeatureValue(lexeme));
         fcX.Add("root", new FeatureValue(lexeme));
         fcX.Add("cat", new FeatureValue("X"));
         fsc.Add(fcX);
       }
       // END X Category
       retVal[lexeme] = fsc;
     }//end: if(retVal[lexeme] == null)
   }//end: foreach(string lexeme in lexemesToLookup)
  }//end: if(lexemesToLookup.Count > 0)
  return retVal;
}
```

Appendix C

The following table contains a global view of the complete production rule set (a description of the Grammar rules and the relevant Unification rules).

This pretty printer functionality, which we used to obtain the rules below, has been added to the ePOWER Workbench by Ron van Gog as an extra component. It generates a general view of the complete Production rule set by examining the class model (see Chapter 6). Furthermore, the PC-PATR parser can use it, which is an alternative for the parser used within the ePOWER Workbench, at this moment. For more information about the PC-PATR parser, see Unification-based syntactic parser PATR 2004 [30].

The rules for the noun phrase extraction are based on Ron van Gog [37]. Note that some of the rules have been modified by the author.

Production rules for the Noun-Phrase Extraction
1
Rule{} **PP -> PP_1 CONJ_2 PP_3** <PP inResult> = false <PP sem type> = pp_conj <PP_1 sem type> = pp <PP sem conj> = <CONJ_2 sem> <PP sem s1> = <PP_1 sem> <PP sem s2> = <PP_3 sem>
2
Rule{} **VC -> VCI_1** <VC inResult> = false <VCI_1 head mood> = INDICATIVE <VCI_1 head subcat> = MAIN <VC comp> = <VCI_1 head comp> <VC head> = <VCI_1 head> <VC sem finit main> = <VCI_1 sem> <VC sem finit root> = <VCI_1 root>

3
Rule{} **VC -> VCI_1** <VC inResult> = false <VCI_1 head mood> = INDICATIVE <VCI_1 head subcat> = COPULA <VC comp> = <VCI_1 head comp> <VC head> = <VCI_1 head> <VC sem finit main> = <VCI_1 sem> <VC sem finit root> = <VCI_1 root>
4
Rule{} **PP -> PREP_1 PP_2** <PP inResult> = false <PP sem type> = pp2 <PP sem main> = <PP_2 sem> <PP sem prep> = <PREP_1 root>
5
Rule{} **NP -> NP_1 PN_2 VP_3** <NP inResult> = true <NP sem modif type> = bijvoeglijke_bijzin <PN_2 head subcat> = RELATIVE <NP sem modif main> = <VP_3 sem> <NP sem modif pn> = <PN_2 sem> <NP sem> = <NP_1 sem> <NP head agr> = <NP_1 head agr> <NP head agr> = <PN_2 head agr> <NP head agr> = <VP_3 head agr>
6
Rule{} **NP -> NP_1 CONJ_2 NP_3** <NP inResult> = true <NP head agr per> = 3 <NP sem type> = np_conj <NP_1 sem conj> = null <NP sem conj> = <CONJ_2 sem> <NP sem s1> = <NP_1 sem> <NP sem s2> = <NP_3 sem>

7
Rule{} **PP -> PREP_1 NP_2** <PP inResult> = false <PP sem type> = pp <PP sem main> = <NP_2 sem> <PP sem prep> = <PREP_1 root>
8
Rule{} **VP -> (ADJP_1) VC_2** <VP inResult> = false <VP sem type> = vp <VC_2 head subcat> = COPULA <VP head> = <VC_2 head> <VP sem adj> = <ADJP_1 sem> <VP sem pred> = <VC_2 sem>
9
Rule{} **NP -> NP_1 PN_2 XLIST_3 PUNCT_4** <NP inResult> = true <NP sem modif type> = bijvoeglijke_bijzin <PN_2 head subcat> = RELATIVE <NP sem modif main> = <XLIST_3 sem> <NP sem modif pn> = <PN_2 sem> <NP sem> = <NP_1 sem> <NP head agr> = <NP_1 head agr> <NP head agr> = <PN_2 head agr>
10
Rule{} **NP -> NP_1 ADV_2 (NP_3) XLIST_4 PUNCT_5** <NP inResult> = true <NP sem modif type> = nabepaling <ADV_2 head subcat> = RELATIVE <NP sem modif main> = <XLIST_4 sem> <NP sem modif adv> = <ADV_2 sem> <NP sem modif np> = <NP_3 sem> <NP sem> = <NP_1 sem> <NP head agr> = <NP_1 head agr> <NP head agr> = <ADV_2 head agr>

11
Rule{} **VP -> (ADVP_1) VC_2** <VP inResult> = false <VP sem type> = vp <VP head> = <VC_2 head> <VP sem adv> = <ADVP_1 sem> <VP sem pred> = <VC_2 sem>
12
Rule{} **VC -> VCI_1 VCI_2** <VC inResult> = false <VCI_1 head mood> = INDICATIVE <VCI_1 head subcat> = AUX <VCI_2 head subcat> = MAIN <VC comp> = <VCI_2 head comp> <VC head> = <VCI_1 head> <VC sem finit main> = <VCI_1 sem> <VC sem finit root> = <VCI_1 root> <VC sem hoofd main> = <VCI_2 main> <VC sem hoofd root> = <VCI_2 root> <VCI_1 head needs> = <VCI_2 head mood>
13
Rule{} **VC -> VCI_1 VCI_2** <VC inResult> = false <VCI_1 head mood> = INDICATIVE <VCI_1 head subcat> = AUX <VCI_2 head subcat> = COPULA <VC comp> = <VCI_2 head comp> <VC head> = <VCI_1 head> <VC sem finit main> = <VCI_1 sem> <VC sem finit root> = <VCI_1 root> <VC sem hoofd main> = <VCI_2 sem> <VC sem hoofd root> = <VCI_2 root> <VCI_1 head needs> = <VCI_2 head mood>
14
Rule{} **VCI -> V_1** <VCI inResult> = false <VCI head> = <V_1 head> <VCI root> = <V_1 root> <VCI sem> = <V_1 sem>

15
Rule{} **NP -> A_1** <NP inResult> = true <NP sem type> = np_ref <NP sem isValue> = false <NP sem main> = <A_1 sem> <NP sem href> = <A_1 href>
16
Rule{} **ADVP -> ADV_1** <ADVP inResult> = false <ADVP sem hd type> = adv <ADVP sem type> = adv_list <ADVP sem hd main> = <ADV_1 sem>
17
Rule{} **ADVP -> PP_1** <ADVP inResult> = false <ADVP sem type> = adv_list <ADVP sem hd> = <PP_1 sem>
18
Rule{} **ADVP -> ADVP_1 ADVP_2** <ADVP inResult> = false <ADVP list> = true <ADVP sem type> = adv_list <ADVP_1 sem hd type> = adv <ADVP_2 list> = false <ADVP sem hd> = <ADVP_2 sem hd> <ADVP sem tl> = <ADVP_1 sem>
19
Rule{} **ADJP -> ADJP_1 ADJP_2** <ADJP inResult> = false <ADJP list> = true <ADJP sem type> = adj_list <ADJP_2 list> = false <ADJP sem hd> = <ADJP_2 sem> <ADJP sem tl> = <ADJP_1 sem>

20
Rule{} **ADJP -> (ADVP_1) NUM_2** <ADJP inResult> = false <ADJP sem type> = adj <NUM_2 head subcat> = ORDINAL <ADJP sem adv> = <ADVP_1 sem> <ADJP sem main> = <NUM_2 sem> <ADJP sem root> = <NUM_2 root>
21
Rule{} **ADJP -> (ADVP_1) ADJ_2** <ADJP inResult> = false <ADJP sem type> = adj <ADJP sem adv> = <ADVP_1 sem> <ADJP sem main> = <ADJ_2 sem> <ADJP sem root> = <ADJ_2 root>
22
Rule{} **ADJP -> (ADVP_1) ADJP_2 CONJ_3 ADJP_4** <ADJP inResult> = false <ADJP sem type> = adj_conj <ADJP_2 list> = false <ADJP_2 sem conj> = null <ADJP_4 list> = false <ADJP sem adv> = <ADVP_1 sem> <ADJP sem conj> = <CONJ_3 sem> <ADJP sem s1> = <ADJP_2 sem> <ADJP sem s2> = <ADJP_4 sem>
23
Rule{} **N -> NUM_1** <N inResult> = false <N head agr gen> = MF <N isValue> = true <N head agr> = <NUM_1 head agr> <N root> = <NUM_1 root> <N sem> = <NUM_1 sem>

24
Rule{} **NP -> CUR_1 NUM_2** <NP inResult> = true <NP sem type> = np_money <NP sem isValue> = true <NUM_2 subcat> = CARDINAL <NP sem main> = <NUM_2 sem> <NP sem root> = <NUM_2 root> <NP sem cur> = <CUR_1 sem>
25
Rule{} **NP -> (DETE_1) (ADJP_2) N_3 (PP_4)** <NP inResult> = true <NP head agr per> = 3 <NP sem type> = np <NP head> = <DETE_1 head> <NP head> = <N_3 head> <NP sem adj> = <ADJP_2 sem> <NP sem det> = <DETE_1 sem> <NP sem main> = <N_3 sem> <NP sem ntype> = <N_3 ntype> <NP sem pp> = <PP_4 sem> <NP sem root> = <N_3 root> <NP sem isValue> = <N_3 isValue>
26
Rule{} **N -> V_1** <N inResult> = false <N head agr gen> = N <N head agr num> = S <N ntype> = V <N isValue> = false <V_1 head mood> = INFINITIVE <V_1 head subcat> = MAIN <N root> = <V_1 root> <N sem> = <V_1 sem>
28
Rule{} **DETE -> DET_1** <DETE inResult> = false <DETE head> = <DET_1 head> <DETE sem> = <DET_1 sem>

29
Rule{} **DETE -> PN_1** <DETE inResult> = false <PN_1 head subcat> = {PERSONAL, RELATIVE} <PN_1 head agr case> = C2 <DETE sem> = <PN_1 sem> <DETE head> = <PN_1 head>
30
Rule{} **DETE -> NUM_1** <DETE inResult> = false <NUM_1 head subcat> = CARDINAL <DETE head> = <NUM_1 head> <DETE sem> = <NUM_1 sem>

Production rules for the Verb-Phrase Extraction
1
Rule{(Application of another source)} **PREP -> PREP_1 N_2 PREP_3** <PREP inResult> = true <PREP_1 root> = {op,bij} <N_2 root> = {grond,toepassing} <PREP_3 root> = van <PREP sem> = <PREP root> <PREP sem> = <N_2 root>
2
Rule{(Application Provision)} **S -> NP_1 V_2 (ADV_3) PREP_4 N_5** <S inResult> = true <S sem type> = s_app <V_2 root> = zijn <V_2 head subcat> = MAIN <ADV_3 root> = niet <PREP_4 root> = van <N_5 root> = toepassing <S sem ref> = <NP_1 sem> <S sem adv> = <ADV_3 sem> <NP_1 head agr> = <V_2 head agr>
3
Rule{(Deeming Provision s_order=sv)} **S -> NP_1 V_2 (XLIST_3) V_4 XLIST_5** <S inResult> = true <S sem type> = s_dp <S s_order> = sv <V_2 root> = worden <V_2 head subcat> = AUX <V_4 root> = achten <V_4 head subcat> = MAIN <S head agr> = <NP_1 head agr> <S head agr> = <V_2 head agr> <S sem subject> = <NP_1 sem> <S sem dp_part1> = <V_2 sem> <S sem time_period> = <XLIST_3 sem> <S sem dp_part2> = <V_4 sem> <S sem fiction> = <XLIST_5 sem>

4
Rule{(Deeming Provision s_order=vs)} **S -> V_1 NP_2 (XLIST_3) V_4 XLIST_5** <S inResult> = false <S sem type> = s_dp <S s_order> = vs <V_1 root> = worden <V_1 head subcat> = AUX <V_4 root> = achten <V_4 head mood> = PARTICIPLE <S head agr> = <V_1 head agr> <S head agr> = <NP_2 head agr> <S sem dp_part1> = <V_1 sem> <S sem subject> = <NP_2 sem> <S sem time_period> = <XLIST_3 sem> <S sem dp_part2> = <V_4 sem> <S sem fiction> = <XLIST_5 sem>
5
Rule{(Definition 1)} **S -> NP_1 V_2 NP_3** <S inResult> = true <S sem type> = s_def <V_2 root> = zijn <V_2 head subcat> = MAIN <NP_3 sem isValue> = false <S head agr> = <NP_1 head agr> <S head agr> = <V_2 head agr> <S head> = <V_2 head> <S sem subject> = <NP_1 sem> <S sem direct_object> = <NP_3 sem>

6
Rule{(Definition 2 s_order=sv)} **S -> PREP_1 NP_2 V_3 (ADV_4) V_5 NP_6** <S inResult> = true <S sem type> = s_def2 <S s_order> = sv <PREP_1 root> = onder <V_3 root> = worden <V_3 head subcat> = AUX <V_3 head agr per> = 3 <ADV_4 root> = {mede, niet} <V_5 root> = verstaan <V_5 head mood> = PARTICIPLE <S sem subject> = <NP_2 sem> <S sem definition> = <NP_6 sem> <S sem adv> = <ADV_4 sem>
7
Rule{(Definition 2 s_order=vs)} **S -> V_1 (ADV_2) PREP_3 NP_4 (ADV_5) V_6 NP_7** <S inResult> = false <S sem type> = s_def2 <S s_order> = vs <V_1 root> = worden <V_1 head subcat> = AUX <V_1 head agr per> = 3 <ADV_2 root> = {mede,niet} <PREP_3 root> = onder <ADV_5 root> = {mede, niet} <V_6 root> = verstaan <V_6 head mood> = PARTICIPLE <V_6 head agr per> = 3 <S sem subject> = <NP_4 sem> <S sem definition> = <NP_7 sem> <S sem adv> = <ADV_2 sem> <S sem adv> = <ADV_5 sem>

8
Rule{(Definition 2 s_order=vs)} **S -> V_1 (ADV_2) V_3 PREP_4 NP_5 NP_6** <S inResult> = false <S sem type> = s_def2 <S s_order> = vs <V_1 root> = worden <V_1 head subcat> = AUX <V_1 head agr per> = 3 <ADV_2 root> = {mede, niet} <V_3 root> = verstaan <V_3 head mood> = PARTICIPLE <V_3 head agr per> = 3 <PREP_4 root> = onder <S sem subject> = <NP_5 sem> <S sem definition> = <NP_6 sem> <S sem adv> = <ADV_2 sem>
9
Rule{(Definition 3 s_order=sv)} **S -> PREP_1 NP_2 V_3 (ADV_4) NP_5** <S inResult> = true <S sem type> = s_def3 <S s_order> = sv <PREP_1 root> = tot <V_3 root> = behoren <V_3 head subcat> = MAIN <ADV_4 root> = {mede,niet} <S sem subject> = <NP_2 sem> <S sem adv> = <ADV_4 sem> <S sem definition> = <NP_5 sem> <V_3 head agr> = <NP_5 head agr>

10
Rule{(Definition 3 s_order=vs)} **S -> V_1 (ADV_2) PREP_3 NP_4 (ADV_5) NP_6** <S inResult> = false <S sem type> = s_def3 <S s_order> = vs <V_1 root> = behoren <V_1 head subcat> = MAIN <ADV_2 root> = {mede,niet} <PREP_3 root> = tot <ADV_5 root> = {mede,niet} <S sem subject> = <NP_4 sem> <S sem adv> = <ADV_2 sem> <S sem adv> = <ADV_5 sem> <S sem definition> = <NP_6 sem> <V_1 head agr> = <NP_6 head agr>
11
Rule{(Definition 4 s_order=sv)} **S -> NP_1 V_2 (SDEF_3) V_4 X_5 NP_6** <S inResult> = true <S sem type> = s_def4 <S s_order> = sv <V_2 root > = worden <V_2 head subcat> = AUX <V_2 head agr per> = 3 <V_4 root> = {gelijkstellen,aanmerken} <V_4 head mood> = PARTICIPLE <V_4 head subcat> = MAIN <X_5 sem> = {met,als} <S sem subject> = <NP_6 sem> <S sem definition> = <NP_1 sem> <S sem sdef> = <SDEF_3 sem>

12
Rule{(Definition 4 s_order=vs)} **S -> V_1 (SDEF_2) NP_3 V_4 X_5 NP_6** <S inResult> = true <S sem type> = s_def4 <S s_order> = vs <V_1 root > = worden <V_1 head subcat> = AUX <V_1 head agr per> = 3 <V_4 root> = {aanmerken,gelijkstellen} <V_4 head subcat> = MAIN <V_4 head mood> = PARTICIPLE <X_5 sem> = {met,als} <S sem subject> = <NP_6 sem> <S sem definition> = <NP_3 sem> <S sem sdef> = <SDEF_2 sem>
13
Rule{(Explicit Condition)} **EC -> CONJ_1 NP_2 XLIST_3** <EC inResult> = false <EC sem type> = ec <CONJ_1 root> = {indien, voorzover} <EC sem subject> = <NP_2 sem> <EC sem feature> = <XLIST_3 sem>
14
Rule{(Formula)} **NP -> NP_1 V_2 PREP_3 NP_4** <NP inResult> = false <NP sem type> = np_formula <NP sem isValue> = true <NP_1 sem isValue> = true <V_2 root> = {verminderen,vermeerderen} <V_2 head mood> = PARTICIPLE <PREP_3 root> = met <NP_4 sem isValue> = true <NP sem x> = <NP_1 sem> <NP sem y> = <NP_4 sem> <NP sem plusminus> = <V_2 root>

15
Rule{(Main Sentence)} **S -> SDEF_1 S_2** <S inResult> = true <S_2 s_order> = vs <S head agr> = <S_2 head agr> <S sem sdef> = <SDEF_1 sem> <S sem> = <S_2 sem>
16
Rule{(Main Sentence)} **S_DELETED -> S_XXX_1 (PUNCT_2) CONJ_3 XLIST_4** <S_DELETED inResult> = false <S_XXX_1 s_order> = sv <PUNCT_2 root> = , <S_DELETED head agr> = <S_XXX_1 head agr> <S_DELETED sem bijzin> = <XLIST_4 sem> <S_DELETED sem main> = <S_XXX_1 sem> <S_DELETED sem adv> = <CONJ_3 sem>
17
Rule{(Main sentence)} **S -> EC_1 PUNCT_2 (ADV_3) S_4** <S inResult> = true <PUNCT_2 root> = , <ADV_3 root> = dan <S_4 s_order> = vs <S head agr> = <S_4 head agr> <S sem ec> = <EC_1 sem> <S sem> = <S_4 sem>
18
Rule{(Main Sentence)} **S -> S_1 (PUNCT_2) EC_3** <S inResult> = true <S_1 s_order> = sv <PUNCT_2 root> = , <S head agr> = <S_1 head agr> <S sem ec> = <EC_3 sem> <S sem> = <S_1 sem>

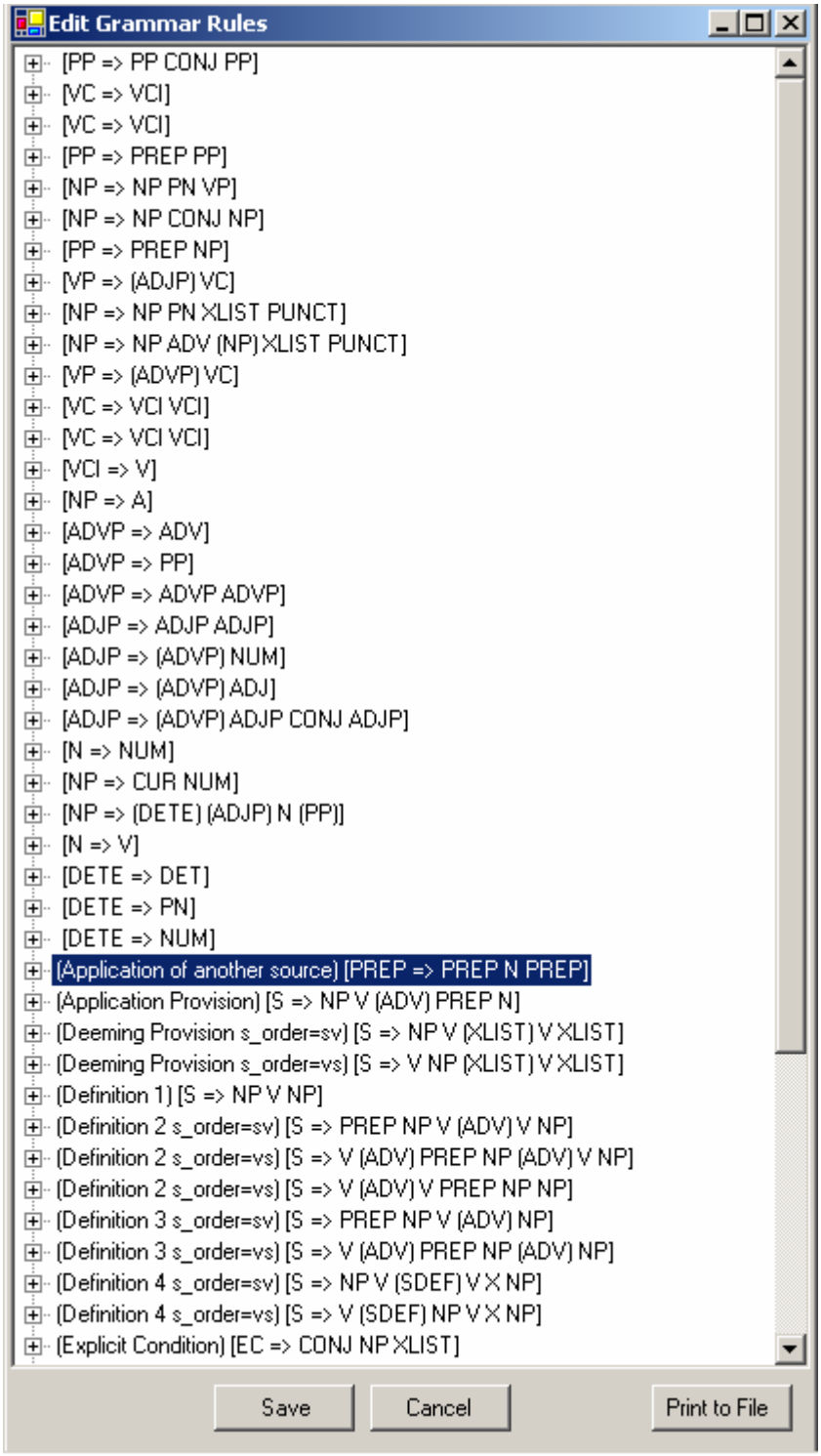
19
Rule{(References)} **PREP -> (CONJ_1) V_2 PREP_3** <CONJ_1 root> = als <V_2 root> = bedoelen <V_2 head mood> = PARTICIPLE <PREP_3 root> = in <PREP sem> = <PREP root> <PREP sem> = <V_2 root>
20
Rule{(Relations)} **S -> NP_1 V_2 PP_3** <S inResult> = true <S sem type> = s_rel <NP_1 head agr case> = C1 <V_2 root> = gelden <PP_3 sem prep> = voor <S sem subject> = <NP_1 sem> <S sem pp> = <PP_3 sem> <S sem verb> = <V_2 sem> <NP_1 head agr> = <V_2 head agr>
21
Rule{(Scope Definition)} **SDEF -> PREP_1 (DET_2) N_3 PREP_4 NP_5** <SDEF inResult> = false <SDEF sem type> = scopedef <PREP_1 root> = voor <DET_2 root> = de <N_3 root> = toepassing <PREP_4 root> = van <NP_5 sem type> = np_ref <SDEF sem ref> = <NP_5 sem> <DET_2 head agr> = <N_3 head agr>

22
Rule{(Value Assignment)} **S -> NP_1 V_2 (N_3) (PREP_4) NP_5** <S inResult> = true <S sem type> = s_va <NP_1 sem isValue> = true <V_2 root> = {zijn,bedragen} <V_2 head subcat> = MAIN <N_3 root> = gelijk <PREP_4 root> = aan <NP_5 sem isValue> = true <S sem subject> = <NP_1 sem> <S sem formula> = <NP_1 head agr> <S sem formula> = <V_2 head agr> <S sem formula> = <NP_5 sem>
23
Rule{(Value Assignment)} **S -> NP_1 V_2 PREP_3 NP_4 V_5** <S inResult> = true <S sem type> = s_va <NP_1 sem isValue> = true <V_2 root> = worden <V_2 head subcat> = AUX <PREP_3 root> = op <NP_4 sem isValue> = true <V_5 root> = stellen <V_5 head mood> = PARTICIPLE <S sem subject> = <NP_1 sem> <S sem formula> = <NP_1 head agr> <S sem formula> = <V_2 head agr> <S sem formula> = <NP_4 sem>
24
Rule{(Xlist)} **XLIST -> X_1 (XLIST_2)** <XLIST inResult> = false <XLIST sem type> = x_list <XLIST sem hd> = <X_1 sem> <XLIST sem tl> = <XLIST_2 sem>

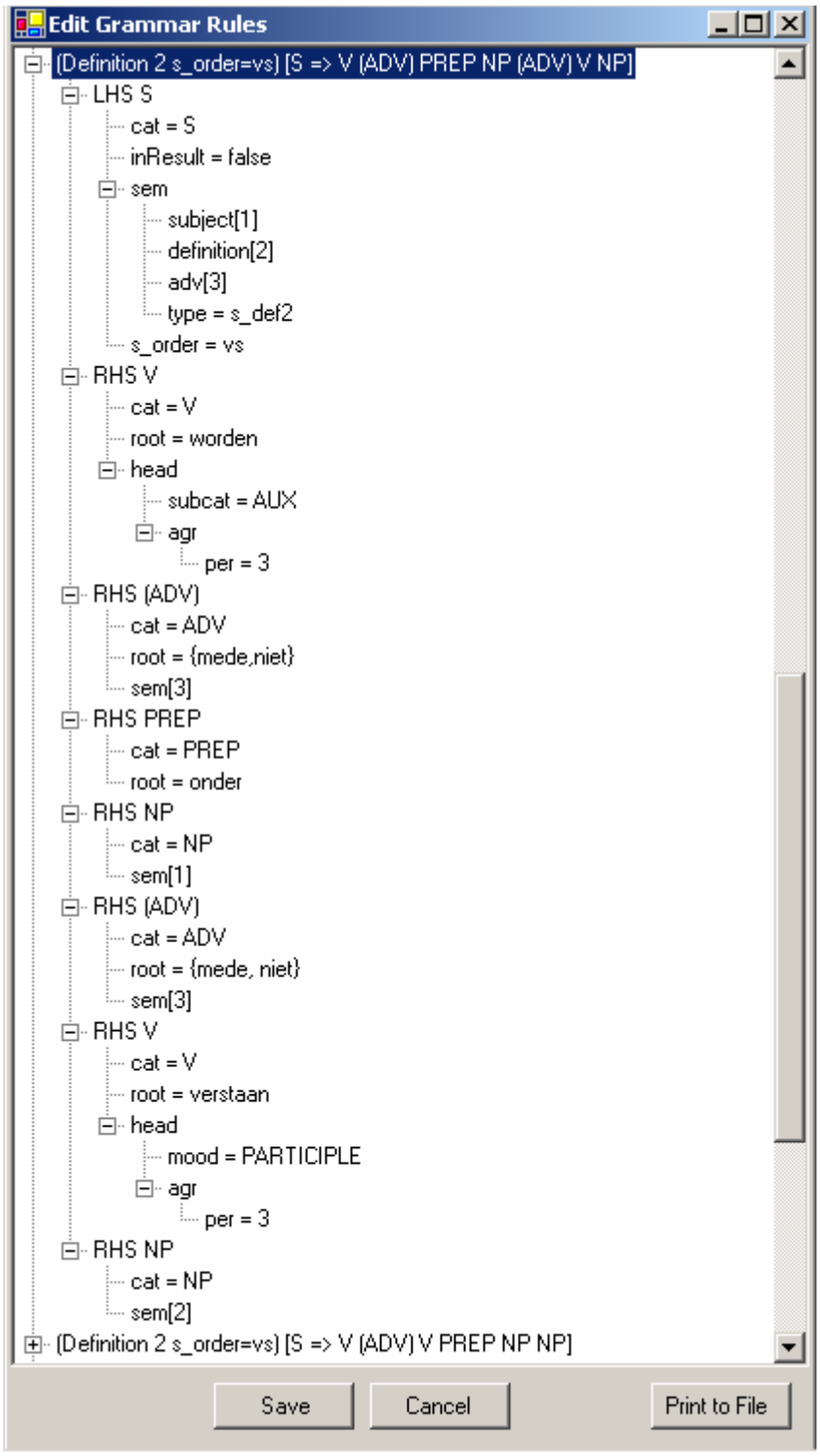
25
Rule{NP} **NP -> NP_1 NP_2** <NP inResult> = true <NP sem root> = Koninkrijk der Nederlanden <NP sem type> = np <NP_1 sem root> = koninkrijk <NP_2 sem root> = Nederland <NP_2 head agr case> = C2

Appendix D

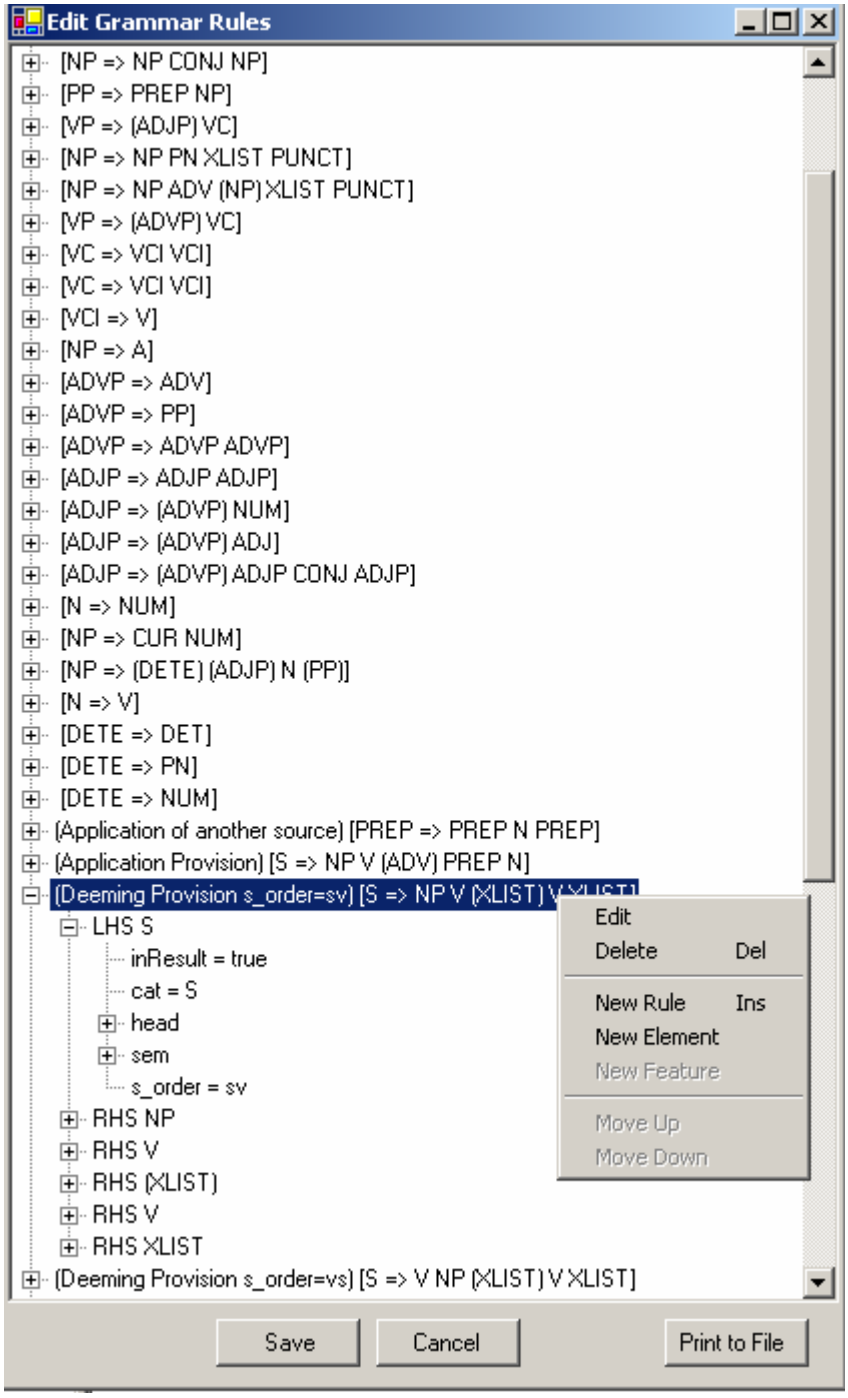
This Appendix contains some screenshots of the *Automated Rule Management* tool (also known as the *Grammar Editor*). Within these screenshots, the access-button ("*Print to File*") for the Printer is also depicted.



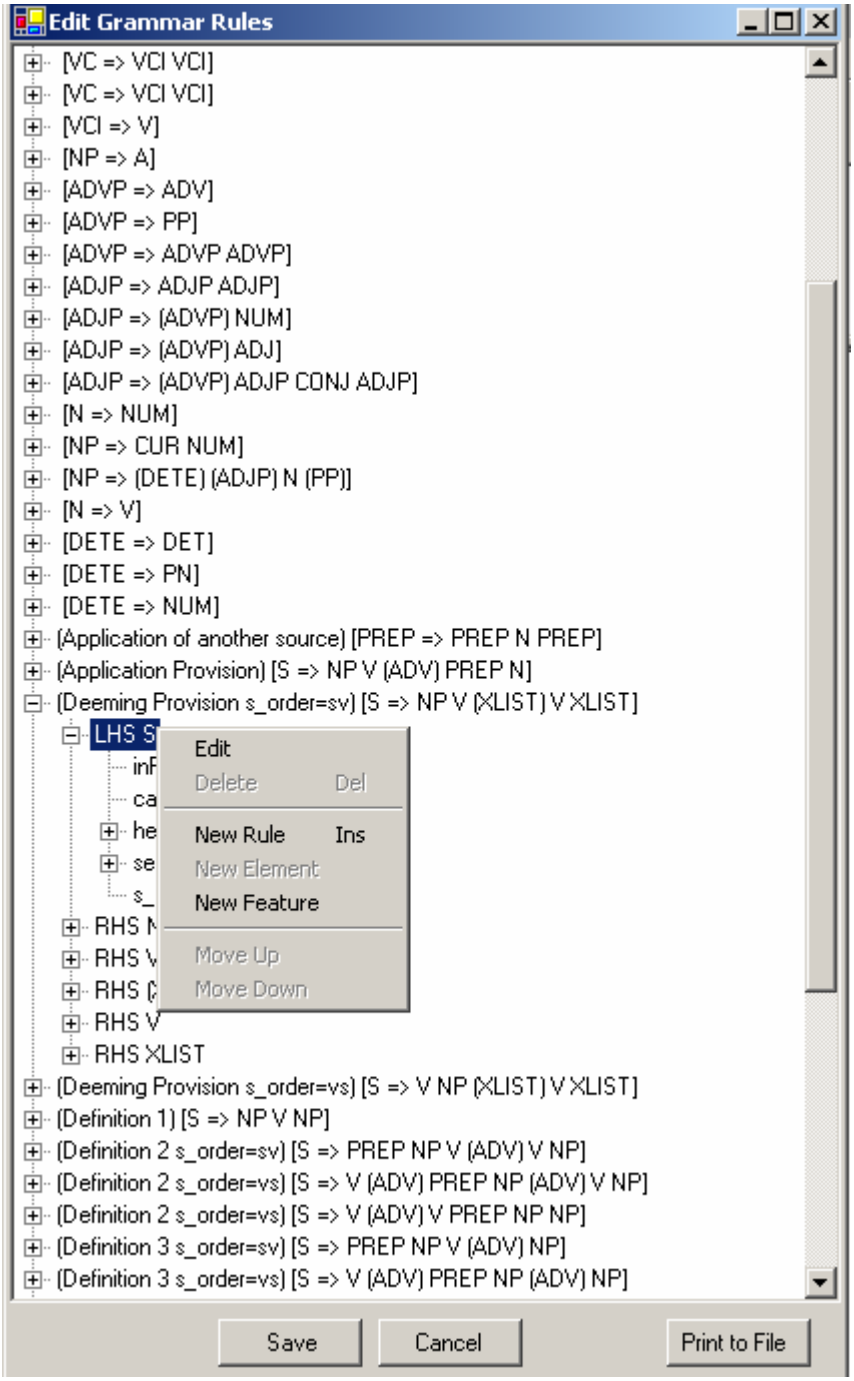
The Grammar Editor with all the Production rules (Grammar- and Unification rules) in a tree like structure.



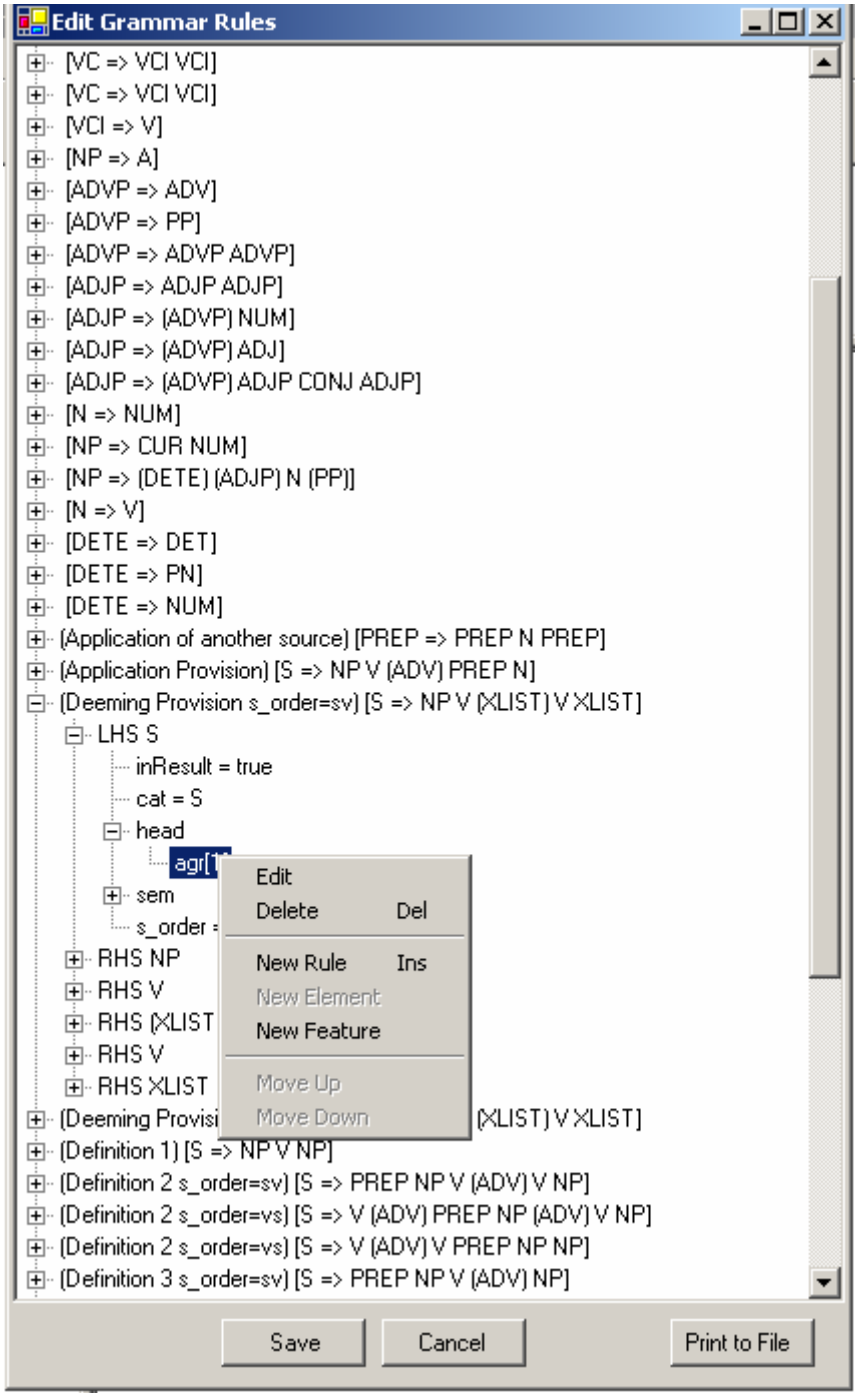
One of the collapsed (tree) nodes for the JLC *Definition* (Definition 2). By this screenshot all the Unification rules are becoming visible, so the grammatical meaning of each of the NLC's (Natural Language Constructs) will become clear.



The possibility to add/edit/delete a Production rule and the possibility to add new NLC's.



The possibility to edit/delete a GrammarElement (NLC) and the possibility to add new GrammarFeatures (Unification Rules) for each of the NLC's.



The possibility to edit/delete a GrammarFeature and the possibility to add new GrammarFeatures (Unification Rules) within a GrammarFeature.

Appendix E

The programming code for the *Automated Rule Management* tool (also known as the *Grammar Editor*).

Workbench.NaturalLanguage.Grammar.Editor.GrammarClassModel.cs

```
//////////////////////////////////////////////////////////////////
/// WORKBENCH                                                   ///
///                                                             ///
///                                                             ///
///                                                             ///
/// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
//////////////////////////////////////////////////////////////////
/// Revision information:                                       ///
///   $Workfile:: GrammarClassModel.cs                        $ ///
///   $Revision:: 1                                           $ ///
///   $Author:: Ron_van_gog, Kamal_Sayah                      $ ///
///   $Date:: 24/02/04                                        $ ///
//////////////////////////////////////////////////////////////////

using System;
using System.Collections;
using System.Data;
using System.Data.SqlClient;
using System.Xml;
using Belastingdienst.Utilities;

// Het klassemodel voor het inlezen en koppelen van de productieregels aan een interne
structuur. De //productieregels kunnen gedurende het programma worden
// gemuteerd en later vanuit dit klassemodel worden teruggeschreven naar de database (ook de
XML //generatie wordt hier beschreven).

namespace Workbench.NaturalLanguage.Grammar.Editor
{
    public class GrammarRuleCollection
    {
// Deze variabele bevat de gegevens voor de connectie naar de benodigde database. Deze gegevens
//worden uit de Windows-registry gehaald.
string connString =
(string)RegistryAndAppSettingsReader.GetSettingValue(@"Belastingdienst\NLP","DbConnectionString");

// De databaseName wordt nu keihard gekoppeld aan de Translate-NL database. Later moet er aan de
//GrammarForm extra functionaliteit worden
// toegevoegd, zodat de gebruiker kan kiezen tussen verschillende talen.
public string databaseName = "Translate-nl";

// Deze klasse zorgt voor de correcte sortering van de knopen in de TreeStructure
public class mySort : IComparer
{
      int IComparer.Compare( Object x, Object y )
      {
            return( (new CaseInsensitiveComparer()).Compare( ((GrammarRule)x).Name,
            ((GrammarRule)y).Name ) );
      }
}

      // De collectie van alle productieregels
      public ArrayList RuleCollection = new ArrayList();

      public GrammarRuleCollection()
      {
      }

// Deze constructor krijgt de datatable van de GrammarForm.GrammarForm_Load methode en vult de
//interne
// productieregelcollectie (en sorteert deze).
      public GrammarRuleCollection(DataTable dt)
      {
            foreach(DataRow row in dt.Rows)
            {
                  GrammarRule rule = new GrammarRule(row);
                  RuleCollection.Add(rule);
            }
            Sort();
      }

      // Het toevoegen van een productieregel
      public void Add(object obj)
      {
```

```
                RuleCollection.Add(obj);
                Sort();
        }

        // Het sorteren van de productieregels
        public void Sort()
        {
                IComparer myComparer = new mySort();
                RuleCollection.Sort(myComparer);
        }

    // In deze methode wordt er een nieuwe connectie gemaakt met de database en wordt de complete
    //(gemuteerde) productieregelset
    // weggeschreven.
        public void Save()
        {
                SqlConnection dbConnection = null;
                SqlCommand dbCommand = null;
                SqlDataAdapter dbDataAdapter = null;

                DataSet dsRules = new DataSet();
                string queryString = "SELECT * FROM ProductionRules";
                String fullQueryString = "use ["+databaseName+"]; "+queryString;

                try
                {
                        dbConnection = new SqlConnection(connString);
                        dbConnection.Open();
                        dbCommand = new SqlCommand(fullQueryString,(SqlConnection)dbConnection);
                        dbDataAdapter = new SqlDataAdapter();
                        dbDataAdapter.SelectCommand = dbCommand;
                        dbDataAdapter.Fill(dsRules);
                        DataTable tblRules = dsRules.Tables[0];

                        ArrayList deletedItems = new ArrayList();
                        foreach (GrammarRule rule in this.RuleCollection)
                        {
                                rule.Save(tblRules);

                        // Als een regel is verwijderd in de Form dan wordt deze regel nog wel bewaard
                        //in een collectie, zodat
                        // deze later alsnog bij het wegschrijven kan worden verwijderd uit de
                        //RuleCollection
                                if (rule.isDeleted)
                                {
                                        deletedItems.Add(rule);
                                }
                        }
                        foreach (GrammarRule rule in deletedItems)
                        {
                                RuleCollection.Remove(rule);
                        }

                        SqlCommandBuilder builder = new SqlCommandBuilder(dbDataAdapter);
                        dbDataAdapter.Update(dsRules);

                        dsRules = new DataSet();
                        dbDataAdapter = new SqlDataAdapter();
                        dbDataAdapter.SelectCommand = dbCommand;
                        dbDataAdapter.Fill(dsRules);
                        tblRules = dsRules.Tables[0];

                        // Het eerst leegmaken en daarna opnieuw vullen van de RuleCollection
                        RuleCollection.Clear();
                        foreach(DataRow row in tblRules.Rows)
                        {
                                GrammarRule rule = new GrammarRule(row);
                                RuleCollection.Add(rule);
                        }
                        Sort();

                }
                catch(System.Data.SqlClient.SqlException exc)
                {
                        throw; // Er wordt een exceptie gegooid!
                }
                finally
                {
                        dbConnection.Close();
                }
        }

    // Dez methode zorgt voor een zogenaamde Pretty Print van de inhoud van de RuleCollection (in
    //HTML)
        public void Print(string Filename)
        {
                string Result="<HTML><HEAD><TITLE></TITLE><BODY><CENTER><TABLE border=1>";
                int pos = 0;
```

```
162
163                 Result += "<CAPTION><b><FONT face=\"Lucida Sans Unicode\" size=6>Grammar
164                 Rules</FONT></b></CAPTION><BR>";
165                       foreach (GrammarRule rule in RuleCollection)
166                       {
167                             if (pos == 0)
168                             {
169                                   Result += "<TR>";
170                             }
171                       Result += "<TD valign=top><FONT face=\"Lucida Sans Unicode\" size=2>"+ rule.Print() +
172                       "</FONT></TD>";
173                             if (pos == 2)
174                             {
175                                   Result += "</TR>";
176                                   pos = 0;
177                             }
178                             else
179                             {
180                                   pos++;
181                             }
182                       }
183                       if (pos > 0)
184                       {
185                             Result += "</TR>";
186                       }
187                       Result+="</TABLE></CENTER></BODY></HTML>";
188
189                       // Het wegschrijven naar een *.html file
190                       System.IO.StreamWriter sw = new System.IO.StreamWriter(Filename);
191                       sw.Write(Result);
192                       sw.Flush();
193                       sw.Close();
194
195                 // Het starten van een programma om de inhoud van de weggeschreven file te tonen (Internet
196                 //Explorer)
197                       try
198                       {
199                             System.Diagnostics.Process.Start(Filename);
200                       }
201                       catch(Exception e)
202                       {
203                             throw new Exception();
204                       }
205                 }
206           }
207
208           // In deze klasse worden de complete productieregels, dus inclusief de features en de elementen
209           //aangemaakt. Tevens wordt er functionaliteit
210           // toegevoegd zodat de regels kunnen worden vertaald naar het bijbehorende XML-script.
211           public class GrammarRule
212           {
213               public string Name;
214               public string Description;
215               public GrammarElement Lhs;
216               public ArrayList Rhs = new ArrayList();
217               public bool isDeleted = false;
218               public bool isNew = false;
219               public int Id;
220
221               public GrammarRule(string name)
222               {
223                     Name = name;
224                     Lhs = new GrammarElement("new");
225                     Lhs.Lhs = true;
226               }
227               public GrammarRule(DataRow row)
228               {
229                     GrammarElement element;
230                     Id = (int)row["id"];
231                     Name = row["Name"].ToString();
232                     Description = row["Description"].ToString();
233
234                     // Het XML document wordt aangemaakt en gevuld
235                     XmlDocument xmlDummyDoc = new XmlDocument();
236
237                     // Voor elk kind van de LHS (element => FeatureSet) wordt de bijbehorende XML
238                gegenereerd.
239                     xmlDummyDoc.LoadXml(row["LHS"].ToString());
240                     XmlNode xmlRuleLhs = xmlDummyDoc.DocumentElement;
241                     foreach (XmlNode temp in xmlRuleLhs.ChildNodes)
242                     {
243                           if (temp.NodeType == XmlNodeType.Element && temp.Name.Equals("FeatureSet"))
244                           {
245                                 Lhs = new GrammarElement(temp);
246                                 Lhs.Lhs = true;
247                                 break;
248                           }
249                     }
```

```
250             // Voor elk kind van de RHS (element => RHSElement) wordt de bijbehorende XML
251         gegenereerd.
252             xmlDummyDoc.LoadXml(row["RHS"].ToString());
253             XmlNode xmlRuleRhs = xmlDummyDoc.DocumentElement;
254             foreach (XmlNode temp in xmlRuleRhs.ChildNodes)
255             {
256                 if (temp.NodeType == XmlNodeType.Element && temp.Name.Equals("RhsElement"))
257                 {
258                     foreach (XmlNode temp2 in temp.ChildNodes)
259                     {
260                     if (temp2.NodeType == XmlNodeType.Element &&
261                     temp2.Name.Equals("FeatureSet"))
262                         {
263                             element = new GrammarElement(temp2);
264                             element.Lhs = false;
265                             foreach (XmlAttribute attr in temp.Attributes)
266                             {
267                                 if (attr.Name.Equals("nothingAllowed"))
268                                 {
269                                     element.Optional = bool.Parse(attr.Value);
270                                     break;
271                                 }
272                             }
273                             this.Rhs.Add(element);
274                             break;
275                         }
276                     }
277                 }
278             }
279         }
280
281         // De Pretty Print methode
282         public string Print()
283         {
284             string Result = "";
285             Result += "Name: <b>"+ Name +"</b>" + "<BR>" + "Description: " + Description +
286         "<BR>";
287             Result += Lhs.Name + " =>";
288             foreach (GrammarElement element in Rhs)
289             {
290                 if (element.Optional)
291                 {
292                     Result += " (" + element.Name + ")";
293                 }
294                 else
295                 {
296                     Result += " " + element.Name;
297                 }
298             }
299             Result += "<BR>" + Lhs.Print();
300             foreach (GrammarElement element in Rhs)
301             {
302                 Result += element.Print();
303             }
304             Result += "<BR>";
305             return Result;
306         }
307
308         // De methode die het complete XML-script opleverd dat hoort bij de LHS van elke
309         productieregel
310         public string createLhsXml()
311         {
312             XmlDocument doc = new XmlDocument();
313             XmlNode node = doc.CreateElement("Lhs");
314             node.AppendChild(this.Lhs.createXml(doc));
315             return node.OuterXml;
316         }
317
318         // De methode die het complete XML-script opleverd dat hoort bij de RHS van elke
319         productieregel
320         public string createRhsXml()
321         {
322             XmlDocument doc = new XmlDocument();
323             XmlNode node = doc.CreateElement("Rhs");
324             foreach (GrammarElement element in this.Rhs)
325             {
326                 // De vaste constructen in het XML-script toevoegen
327                 XmlNode temp = doc.CreateElement("RhsElement");
328                 XmlAttribute attr = doc.CreateAttribute("nothingAllowed");
329                 attr.Value = element.Optional.ToString().ToLower();
330                 temp.Attributes.Append(attr);
331                 temp.AppendChild(element.createXml(doc));
332                 node.AppendChild(temp);
333             }
334             return node.OuterXml;
335         }
336
337         // Deze methode oveschrijft de gemuteerde regels in de database
```

```
        public void Save(DataTable tblRules)
        {
               DataRow row;

               // Nieuw record in database en gegevens in record schrijven
               // Het id van het nieuwe record in onze class opnemen.
               if (isNew)
               {
                      row = tblRules.NewRow();
                      row["Name"] = Name;
                      row["Description"] = Description;
                      row["LHS"] = createLhsXml();
                      row["RHS"] = createRhsXml();
                      tblRules.Rows.Add(row);
               }
               else if (isDeleted)
               {
                      // Record verwijderen uit database
                      DataRow[] rows = tblRules.Select("id="+Id.ToString());
                      row = rows[0];
                      row.Delete();
               }
               else
               {
                      // Record opzoeken in database en gegevens overschrijven
                      DataRow[] rows = tblRules.Select("id="+Id.ToString());
                      row = rows[0];
                      row["Name"] = Name;
                      row["Description"] = Description;
                      row["LHS"] = createLhsXml();
                      row["RHS"] = createRhsXml();
               }
        }
    }

    // Deze klasse slaat alle informatie op die nodig is voor een GrammarElement (in het
    klassemodel)
    public class GrammarElement
    {
        public string _name;
        public bool Optional=false;
        public bool Lhs=false;
        public ArrayList FeatureSet = new ArrayList();
        public GrammarElement(string name)
        {
               _name = name;
               GrammarFeature f = new GrammarFeature("cat");
               f.Atomic = true;
               f.Fixed = true;
               f.FeatureValue = name;
               FeatureSet.Add(f);
        }
        public GrammarElement(XmlNode node) // Verwacht een <FeatureSet> tag
        {
               GrammarFeature feature;
               foreach(XmlNode temp in node)
               {
                      if (temp.NodeType == XmlNodeType.Element && temp.Name.Equals("Feature"))
                      {
                             feature = new GrammarFeature(temp);
                             if (feature.Name.Equals("cat"))
                             {
                                    this._name = feature.FeatureValue;
                             }
                             this.FeatureSet.Add(feature);
                      }
               }
        }

        public string Name
        {
               get
               {
                      return _name;
               }
               set
               {
                      _name = value;
                      foreach (GrammarFeature f in FeatureSet)
                      {
                             if (f.Name.Equals("cat"))
                             {
                                    f.FeatureValue = value;
                             }
                      }
               }
        }
```

```
426             // De Pretty Print methode
427             public string Print()
428             {
429                     string Result = "<I>" + Name + "</I>" + "<BR>";
430                     foreach (GrammarFeature feature in FeatureSet)
431                     {
432                             Result += feature.Print(1);
433                     }
434                     return Result;
435             }
436
437             // De methode die de XMLNode retourneert van elk GrammarElement
438             public XmlNode createXml(XmlDocument doc)
439             {
440                     XmlNode node = doc.CreateElement("FeatureSet");
441                     foreach (object temp in this.FeatureSet)
442                     {
443                             node.AppendChild(((GrammarFeature)temp).createXml(doc));
444                     }
445                     return node;
446             }
447     }
448
449     // Deze klasse slaat alle informatie op die nodig is voor een GrammarFeature (in het
450     klassemodel)
451     public class GrammarFeature
452     {
453         public string Name;
454         public string FeatureValue;
455         public int EquationId = -1;
456         public bool Atomic = false;
457         public bool Fixed = false;
458         public ArrayList FeatureSet = new ArrayList();
459         public GrammarFeature(string name)
460         {
461                 Name = name;
462         }
463         public GrammarFeature(XmlNode node) // Verwacht een <Feature> tag
464         {
465                 foreach (XmlAttribute attr in node.Attributes)
466                 {
467                         if (attr.Name.Equals("name"))
468                         {
469                                 this.Name = attr.Value;
470                         }
471                         else if (attr.Name.Equals("equationId"))
472                         {
473                                 this.EquationId = int.Parse(attr.Value);
474                         }
475                 }
476                 foreach (XmlNode temp in node.ChildNodes)
477                 {
478                         if (temp.NodeType == XmlNodeType.Element && temp.Name.Equals("AtomicValue"))
479                         {
480                                 this.Atomic = true;
481                                 this.FeatureValue = temp.InnerText;
482                                 break;
483                         }
484                         if (temp.NodeType == XmlNodeType.Element && temp.Name.Equals("ComplexValue"))
485                         {
486                                 GrammarFeature feature;
487                                 this.Atomic = false;
488                                 foreach (XmlNode temp2 in temp.ChildNodes)
489                                 {
490                                 if (temp2.NodeType == XmlNodeType.Element &&
491                                 temp2.Name.Equals("Feature"))
492                                         {
493                                                 feature = new GrammarFeature(temp2);
494                                                 if (feature.Name.Equals("cat"))
495                                                 {
496                                                         this.Name = feature.FeatureValue;
497                                                 }
498                                                 this.FeatureSet.Add(feature);
499                                         }
500                                 }
501                                 break;
502                         }
503                 }
504         }
505
506         // De Pretty Print methode
507         public string Print(int Indent)
508         {
509                 string Result = "";
510                 for (int x = 0; x < Indent ; x++)
511                 {
512                         Result += " ";
513                 }
```

```
514                     Result += "<B>" + Name + "</B>";
515                     if (EquationId > -1)
516                     {
517                             Result +=     "[" + EquationId.ToString() + "]";
518                     }
519                     if (Atomic)
520                     {
521                             Result += " = " + FeatureValue + "<BR>";
522                     }
523                     else
524                     {
525                             Result += "<BR>";
526                             foreach (GrammarFeature feature in FeatureSet)
527                             {
528                                     Result += feature.Print(Indent+1);
529                             }
530                     }
531                     return Result;
532             }
533
534             // De methode die de XMLNode retourneert van elk GrammarFeature
535             public XmlNode createXml(XmlDocument doc)
536             {
537                     XmlNode node = doc.CreateElement("Feature");
538                     XmlNode temp;
539                     XmlAttribute attr;
540                     attr = doc.CreateAttribute("name");
541                     attr.Value = this.Name;
542                     node.Attributes.Append(attr);
543                     if (this.EquationId > -1)
544                     {
545                             attr = doc.CreateAttribute("equationId");
546                     attr.Value = this.EquationId.ToString(); // deze heeft standaard waarde "false", dus
547                     // gewoon toString methode aanroepen voor de waarde
548                             node.Attributes.Append(attr);
549                     }
550                     if (this.Atomic)
551                     {
552                             temp = doc.CreateElement("AtomicValue");
553                             temp.InnerText = this.FeatureValue;
554                             node.AppendChild(temp);
555                     }
556                     else
557                     {
558                             temp = doc.CreateElement("ComplexValue");
559                             foreach (GrammarFeature feature in this.FeatureSet)
560                             {
561                                     temp.AppendChild(feature.createXml(doc));
562                             }
563                             node.AppendChild(temp);
564                     }
565                     return node;
566             }
567              }
568     }
```

Workbench.NaturalLanguage.Grammar.Editor.GrammarElementEdit.cs

```
////////////////////////////////////////////////////////////////////
/// WORKBENCH                                                     ///
///                                                               ///
///                                                               ///
///                                                               ///
/// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
////////////////////////////////////////////////////////////////////
/// Revision information:                                         ///
///   $Workfile:: GrammarElementEdit.cs                         $///
///   $Revision:: 1                                             $///
///   $Author:: Ron_van_gog, Kamal_Sayah                        $///
///   $Date:: 24/02/04                                          $///
////////////////////////////////////////////////////////////////////
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

// Deze klasse omschrijft de complete GrammarElementEdit-form

namespace Workbench.NaturalLanguage.Grammar.Editor
{
   public class GrammarElementEdit : System.Windows.Forms.Form
   {
          private GrammarRule myRule;
          private GrammarElement myElement;
          private bool newElement;
          private System.Windows.Forms.Label label1;
          private System.Windows.Forms.TextBox tbName;
          private System.Windows.Forms.CheckBox cbOptional;
          private System.Windows.Forms.Button cbOk;
          private System.Windows.Forms.Button cbCancel;
          private System.Windows.Forms.Label lbCaptionEdit;
          private System.Windows.Forms.Label lbCaptionNew;
          private System.ComponentModel.Container components = null;

          public GrammarElementEdit()
          {
                 //
                 // Required for Windows Form Designer support
                 //
                 InitializeComponent();

                 //
                 // TODO: Add any constructor code after InitializeComponent call
                 //
          }

          /// <summary>
          /// Clean up any resources being used.
          /// </summary>
          protected override void Dispose( bool disposing )
          {
                 if( disposing )
                 {
                        if(components != null)
                        {
                               components.Dispose();
                        }
                 }
                 base.Dispose( disposing );
          }

          #region Windows Form Designer generated code
          /// <summary>
          /// Required method for Designer support - do not modify
          /// the contents of this method with the code editor.
          /// </summary>
          private void InitializeComponent()
          {
          System.Resources.ResourceManager resources = new
          System.Resources.ResourceManager(typeof(GrammarElementEdit));
          this.label1 = new System.Windows.Forms.Label();
          this.tbName = new System.Windows.Forms.TextBox();
          this.cbOptional = new System.Windows.Forms.CheckBox();
          this.cbOk = new System.Windows.Forms.Button();
          this.cbCancel = new System.Windows.Forms.Button();
          this.lbCaptionEdit = new System.Windows.Forms.Label();
          this.lbCaptionNew = new System.Windows.Forms.Label();
          this.SuspendLayout();
          //
          // label1
          //
```

```
            this.label1.AccessibleDescription =
            resources.GetString("label1.AccessibleDescription");
            this.label1.AccessibleName = resources.GetString("label1.AccessibleName");
            this.label1.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("label1.Anchor")));
            this.label1.AutoSize = ((bool)(resources.GetObject("label1.AutoSize")));
            this.label1.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("label1.Dock")));
            this.label1.Enabled = ((bool)(resources.GetObject("label1.Enabled")));
            this.label1.Font = ((System.Drawing.Font)(resources.GetObject("label1.Font")));
            this.label1.Image = ((System.Drawing.Image)(resources.GetObject("label1.Image")));
            this.label1.ImageAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("label1.ImageAlign")));
            this.label1.ImageIndex = ((int)(resources.GetObject("label1.ImageIndex")));
            this.label1.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("label1.ImeMode")));
            this.label1.Location =
            ((System.Drawing.Point)(resources.GetObject("label1.Location")));
            this.label1.Name = "label1";
            this.label1.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("label1.RightToLeft")));
            this.label1.Size = ((System.Drawing.Size)(resources.GetObject("label1.Size")));
            this.label1.TabIndex = ((int)(resources.GetObject("label1.TabIndex")));
            this.label1.Text = resources.GetString("label1.Text");
            this.label1.TextAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("label1.TextAlign")));
            this.label1.Visible = ((bool)(resources.GetObject("label1.Visible")));
            //
            // tbName
            //
            this.tbName.AccessibleDescription =
            resources.GetString("tbName.AccessibleDescription");
            this.tbName.AccessibleName = resources.GetString("tbName.AccessibleName");
            this.tbName.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("tbName.Anchor")));
            this.tbName.AutoSize = ((bool)(resources.GetObject("tbName.AutoSize")));
            this.tbName.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("tbName.BackgroundImage")));
            this.tbName.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("tbName.Dock")));
            this.tbName.Enabled = ((bool)(resources.GetObject("tbName.Enabled")));
            this.tbName.Font = ((System.Drawing.Font)(resources.GetObject("tbName.Font")));
            this.tbName.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("tbName.ImeMode")));
            this.tbName.Location =
            ((System.Drawing.Point)(resources.GetObject("tbName.Location")));
            this.tbName.MaxLength = ((int)(resources.GetObject("tbName.MaxLength")));
            this.tbName.Multiline = ((bool)(resources.GetObject("tbName.Multiline")));
            this.tbName.Name = "tbName";
            this.tbName.PasswordChar = ((char)(resources.GetObject("tbName.PasswordChar")));
            this.tbName.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("tbName.RightToLeft")));
            this.tbName.ScrollBars =
            ((System.Windows.Forms.ScrollBars)(resources.GetObject("tbName.ScrollBars")));
            this.tbName.Size = ((System.Drawing.Size)(resources.GetObject("tbName.Size")));
            this.tbName.TabIndex = ((int)(resources.GetObject("tbName.TabIndex")));
            this.tbName.Text = resources.GetString("tbName.Text");
            this.tbName.TextAlign =
            ((System.Windows.Forms.HorizontalAlignment)(resources.GetObject("tbName.TextAlign")))
            ;
            this.tbName.Visible = ((bool)(resources.GetObject("tbName.Visible")));
            this.tbName.WordWrap = ((bool)(resources.GetObject("tbName.WordWrap")));
            //
            // cbOptional
            //
            this.cbOptional.AccessibleDescription =
            resources.GetString("cbOptional.AccessibleDescription");
            this.cbOptional.AccessibleName = resources.GetString("cbOptional.AccessibleName");
            this.cbOptional.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbOptional.Anchor")));
            this.cbOptional.Appearance =
            ((System.Windows.Forms.Appearance)(resources.GetObject("cbOptional.Appearance")));
            this.cbOptional.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("cbOptional.BackgroundImage")));
            this.cbOptional.CheckAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbOptional.CheckAlign")));
            this.cbOptional.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("cbOptional.Dock")));
            this.cbOptional.Enabled = ((bool)(resources.GetObject("cbOptional.Enabled")));
            this.cbOptional.FlatStyle =
            ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbOptional.FlatStyle")));
            this.cbOptional.Font =
            ((System.Drawing.Font)(resources.GetObject("cbOptional.Font")));
            this.cbOptional.Image =
            ((System.Drawing.Image)(resources.GetObject("cbOptional.Image")));
            this.cbOptional.ImageAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbOptional.ImageAlign")));
            this.cbOptional.ImageIndex = ((int)(resources.GetObject("cbOptional.ImageIndex")));
```

```
173                     this.cbOptional.ImeMode =
174                     ((System.Windows.Forms.ImeMode)(resources.GetObject("cbOptional.ImeMode")));
175                     this.cbOptional.Location =
176                     ((System.Drawing.Point)(resources.GetObject("cbOptional.Location")));
177                     this.cbOptional.Name = "cbOptional";
178                     this.cbOptional.RightToLeft =
179                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbOptional.RightToLeft")));
180                     this.cbOptional.Size =
181                     ((System.Drawing.Size)(resources.GetObject("cbOptional.Size")));
182                     this.cbOptional.TabIndex = ((int)(resources.GetObject("cbOptional.TabIndex")));
183                     this.cbOptional.Text = resources.GetString("cbOptional.Text");
184                     this.cbOptional.TextAlign =
185                     ((System.Drawing.ContentAlignment)(resources.GetObject("cbOptional.TextAlign")));
186                     this.cbOptional.Visible = ((bool)(resources.GetObject("cbOptional.Visible")));
187                     //
188                     // cbOk
189                     //
190                     this.cbOk.AccessibleDescription = resources.GetString("cbOk.AccessibleDescription");
191                     this.cbOk.AccessibleName = resources.GetString("cbOk.AccessibleName");
192                     this.cbOk.Anchor =
193                     ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbOk.Anchor")));
194                     this.cbOk.BackgroundImage =
195                     ((System.Drawing.Image)(resources.GetObject("cbOk.BackgroundImage")));
196                     this.cbOk.Dock =
197                     ((System.Windows.Forms.DockStyle)(resources.GetObject("cbOk.Dock")));
198                     this.cbOk.Enabled = ((bool)(resources.GetObject("cbOk.Enabled")));
199                     this.cbOk.FlatStyle =
200                     ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbOk.FlatStyle")));
201                     this.cbOk.Font = ((System.Drawing.Font)(resources.GetObject("cbOk.Font")));
202                     this.cbOk.Image = ((System.Drawing.Image)(resources.GetObject("cbOk.Image")));
203                     this.cbOk.ImageAlign =
204                     ((System.Drawing.ContentAlignment)(resources.GetObject("cbOk.ImageAlign")));
205                     this.cbOk.ImageIndex = ((int)(resources.GetObject("cbOk.ImageIndex")));
206                     this.cbOk.ImeMode =
207                     ((System.Windows.Forms.ImeMode)(resources.GetObject("cbOk.ImeMode")));
208                     this.cbOk.Location = ((System.Drawing.Point)(resources.GetObject("cbOk.Location")));
209                     this.cbOk.Name = "cbOk";
210                     this.cbOk.RightToLeft =
211                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbOk.RightToLeft")));
212                     this.cbOk.Size = ((System.Drawing.Size)(resources.GetObject("cbOk.Size")));
213                     this.cbOk.TabIndex = ((int)(resources.GetObject("cbOk.TabIndex")));
214                     this.cbOk.Text = resources.GetString("cbOk.Text");
215                     this.cbOk.TextAlign =
216                     ((System.Drawing.ContentAlignment)(resources.GetObject("cbOk.TextAlign")));
217                     this.cbOk.Visible = ((bool)(resources.GetObject("cbOk.Visible")));
218                     this.cbOk.Click += new System.EventHandler(this.cbOk_Click);
219                     //
220                     // cbCancel
221                     //
222                     this.cbCancel.AccessibleDescription =
223                     resources.GetString("cbCancel.AccessibleDescription");
224                     this.cbCancel.AccessibleName = resources.GetString("cbCancel.AccessibleName");
225                     this.cbCancel.Anchor =
226                     ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbCancel.Anchor")));
227                     this.cbCancel.BackgroundImage =
228                     ((System.Drawing.Image)(resources.GetObject("cbCancel.BackgroundImage")));
229                     this.cbCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;
230                     this.cbCancel.Dock =
231                     ((System.Windows.Forms.DockStyle)(resources.GetObject("cbCancel.Dock")));
232                     this.cbCancel.Enabled = ((bool)(resources.GetObject("cbCancel.Enabled")));
233                     this.cbCancel.FlatStyle =
234                     ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbCancel.FlatStyle")));
235                     this.cbCancel.Font = ((System.Drawing.Font)(resources.GetObject("cbCancel.Font")));
236                     this.cbCancel.Image =
237                     ((System.Drawing.Image)(resources.GetObject("cbCancel.Image")));
238                     this.cbCancel.ImageAlign =
239                     ((System.Drawing.ContentAlignment)(resources.GetObject("cbCancel.ImageAlign")));
240                     this.cbCancel.ImageIndex = ((int)(resources.GetObject("cbCancel.ImageIndex")));
241                     this.cbCancel.ImeMode =
242                     ((System.Windows.Forms.ImeMode)(resources.GetObject("cbCancel.ImeMode")));
243                     this.cbCancel.Location =
244                     ((System.Drawing.Point)(resources.GetObject("cbCancel.Location")));
245                     this.cbCancel.Name = "cbCancel";
246                     this.cbCancel.RightToLeft =
247                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbCancel.RightToLeft")));
248                     this.cbCancel.Size = ((System.Drawing.Size)(resources.GetObject("cbCancel.Size")));
249                     this.cbCancel.TabIndex = ((int)(resources.GetObject("cbCancel.TabIndex")));
250                     this.cbCancel.Text = resources.GetString("cbCancel.Text");
251                     this.cbCancel.TextAlign =
252                     ((System.Drawing.ContentAlignment)(resources.GetObject("cbCancel.TextAlign")));
253                     this.cbCancel.Visible = ((bool)(resources.GetObject("cbCancel.Visible")));
254                     this.cbCancel.Click += new System.EventHandler(this.cbCancel_Click);
255                     //
256                     // lbCaptionEdit
257                     //
258                     this.lbCaptionEdit.AccessibleDescription =
259                     resources.GetString("lbCaptionEdit.AccessibleDescription");
```

```
260                     this.lbCaptionEdit.AccessibleName =
261                     resources.GetString("lbCaptionEdit.AccessibleName");
262                     this.lbCaptionEdit.Anchor =
263                     ((System.Windows.Forms.AnchorStyles)(resources.GetObject("lbCaptionEdit.Anchor")));
264                     this.lbCaptionEdit.AutoSize =
265                     ((bool)(resources.GetObject("lbCaptionEdit.AutoSize")));
266                     this.lbCaptionEdit.Dock =
267                     ((System.Windows.Forms.DockStyle)(resources.GetObject("lbCaptionEdit.Dock")));
268                     this.lbCaptionEdit.Enabled = ((bool)(resources.GetObject("lbCaptionEdit.Enabled")));
269                     this.lbCaptionEdit.Font =
270                     ((System.Drawing.Font)(resources.GetObject("lbCaptionEdit.Font")));
271                     this.lbCaptionEdit.Image =
272                     ((System.Drawing.Image)(resources.GetObject("lbCaptionEdit.Image")));
273                     this.lbCaptionEdit.ImageAlign =
274                     ((System.Drawing.ContentAlignment)(resources.GetObject("lbCaptionEdit.ImageAlign")));
275                     this.lbCaptionEdit.ImageIndex =
276                     ((int)(resources.GetObject("lbCaptionEdit.ImageIndex")));
277                     this.lbCaptionEdit.ImeMode =
278                     ((System.Windows.Forms.ImeMode)(resources.GetObject("lbCaptionEdit.ImeMode")));
279                     this.lbCaptionEdit.Location =
280                     ((System.Drawing.Point)(resources.GetObject("lbCaptionEdit.Location")));
281                     this.lbCaptionEdit.Name = "lbCaptionEdit";
282                     this.lbCaptionEdit.RightToLeft =
283                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("lbCaptionEdit.RightToLeft"))
284                     );
285                     this.lbCaptionEdit.Size =
286                     ((System.Drawing.Size)(resources.GetObject("lbCaptionEdit.Size")));
287                     this.lbCaptionEdit.TabIndex = ((int)(resources.GetObject("lbCaptionEdit.TabIndex")));
288                     this.lbCaptionEdit.Text = resources.GetString("lbCaptionEdit.Text");
289                     this.lbCaptionEdit.TextAlign =
290                     ((System.Drawing.ContentAlignment)(resources.GetObject("lbCaptionEdit.TextAlign")));
291                     this.lbCaptionEdit.Visible = ((bool)(resources.GetObject("lbCaptionEdit.Visible")));
292                     //
293                     // lbCaptionNew
294                     //
295                     this.lbCaptionNew.AccessibleDescription =
296                     resources.GetString("lbCaptionNew.AccessibleDescription");
297                     this.lbCaptionNew.AccessibleName =
298                     resources.GetString("lbCaptionNew.AccessibleName");
299                     this.lbCaptionNew.Anchor =
300                     ((System.Windows.Forms.AnchorStyles)(resources.GetObject("lbCaptionNew.Anchor")));
301                     this.lbCaptionNew.AutoSize = ((bool)(resources.GetObject("lbCaptionNew.AutoSize")));
302                     this.lbCaptionNew.Dock =
303                     ((System.Windows.Forms.DockStyle)(resources.GetObject("lbCaptionNew.Dock")));
304                     this.lbCaptionNew.Enabled = ((bool)(resources.GetObject("lbCaptionNew.Enabled")));
305                     this.lbCaptionNew.Font =
306                     ((System.Drawing.Font)(resources.GetObject("lbCaptionNew.Font")));
307                     this.lbCaptionNew.Image =
308                     ((System.Drawing.Image)(resources.GetObject("lbCaptionNew.Image")));
309                     this.lbCaptionNew.ImageAlign =
310                     ((System.Drawing.ContentAlignment)(resources.GetObject("lbCaptionNew.ImageAlign")));
311                     this.lbCaptionNew.ImageIndex =
312                     ((int)(resources.GetObject("lbCaptionNew.ImageIndex")));
313                     this.lbCaptionNew.ImeMode =
314                     ((System.Windows.Forms.ImeMode)(resources.GetObject("lbCaptionNew.ImeMode")));
315                     this.lbCaptionNew.Location =
316                     ((System.Drawing.Point)(resources.GetObject("lbCaptionNew.Location")));
317                     this.lbCaptionNew.Name = "lbCaptionNew";
318                     this.lbCaptionNew.RightToLeft =
319                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("lbCaptionNew.RightToLeft")))
320                     ;
321                     this.lbCaptionNew.Size =
322                     ((System.Drawing.Size)(resources.GetObject("lbCaptionNew.Size")));
323                     this.lbCaptionNew.TabIndex = ((int)(resources.GetObject("lbCaptionNew.TabIndex")));
324                     this.lbCaptionNew.Text = resources.GetString("lbCaptionNew.Text");
325                     this.lbCaptionNew.TextAlign =
326                     ((System.Drawing.ContentAlignment)(resources.GetObject("lbCaptionNew.TextAlign")));
327                     this.lbCaptionNew.Visible = ((bool)(resources.GetObject("lbCaptionNew.Visible")));
328                     //
329                     // GrammarElementEdit
330                     //
331                     this.AcceptButton = this.cbOk;
332                     this.AccessibleDescription = resources.GetString("$this.AccessibleDescription");
333                     this.AccessibleName = resources.GetString("$this.AccessibleName");
334                     this.AutoScaleBaseSize =
335                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScaleBaseSize")));
336                     this.AutoScroll = ((bool)(resources.GetObject("$this.AutoScroll")));
337                     this.AutoScrollMargin =
338                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScrollMargin")));
339                     this.AutoScrollMinSize =
340                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScrollMinSize")));
341                     this.BackgroundImage =
342                     ((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
343                     this.CancelButton = this.cbCancel;
344                     this.ClientSize = ((System.Drawing.Size)(resources.GetObject("$this.ClientSize")));
345                     this.Controls.Add(this.lbCaptionNew);
346                     this.Controls.Add(this.lbCaptionEdit);
347                     this.Controls.Add(this.cbCancel);
```

```
348                     this.Controls.Add(this.cbOk);
349                     this.Controls.Add(this.cbOptional);
350                     this.Controls.Add(this.tbName);
351                     this.Controls.Add(this.label1);
352                     this.Enabled = ((bool)(resources.GetObject("$this.Enabled")));
353                     this.Font = ((System.Drawing.Font)(resources.GetObject("$this.Font")));
354                     this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
355                     this.ImeMode =
356                     ((System.Windows.Forms.ImeMode)(resources.GetObject("$this.ImeMode")));
357                     this.Location = ((System.Drawing.Point)(resources.GetObject("$this.Location")));
358                     this.MaximumSize = ((System.Drawing.Size)(resources.GetObject("$this.MaximumSize")));
359                     this.MinimumSize = ((System.Drawing.Size)(resources.GetObject("$this.MinimumSize")));
360                     this.Name = "GrammarElementEdit";
361                     this.RightToLeft =
362                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("$this.RightToLeft")));
363                     this.StartPosition =
364                     ((System.Windows.Forms.FormStartPosition)(resources.GetObject("$this.StartPosition"))
365                     );
366                     this.Text = resources.GetString("$this.Text");
367                     this.ResumeLayout(false);
368
369                     }
370                     #endregion
371
372                     private void cbOk_Click(object sender, System.EventArgs e)
373                     {
374                           if (newElement)
375                           {
376                                 myElement = new GrammarElement(tbName.Text);
377                                 myRule.Rhs.Add(myElement);
378                           }
379                           myElement.Name = tbName.Text;
380                           myElement.Optional = cbOptional.Checked;
381                           this.Close();
382                     }
383
384               // Deze methode wordt in de GrammarForm (de hoofdform) aangeroepen om de uiteindelijke edit
385               //te verwerken in het klassemodel
386                     public void DoEdit(GrammarElement item)
387                     {
388                           this.Text = lbCaptionEdit.Text;
389                           this.myElement = item;
390                           this.newElement = false;
391                           this.tbName.Text = item.Name;
392                           this.cbOptional.Checked = item.Optional;
393                           if (item.Lhs)
394                           {
395                                 this.cbOptional.Enabled = false;
396                           }
397                           this.ShowDialog();
398                     }
399
400                     public void DoNew(GrammarRule item)
401                     {
402                           this.Text = lbCaptionEdit.Text;
403                           this.myRule = item;
404                           this.newElement = true;
405                           this.tbName.Text = "";
406                           this.cbOptional.Checked = false;
407                           this.ShowDialog();
408                     }
409                     private void cbCancel_Click(object sender, System.EventArgs e)
410                     {
411                           this.Close();
412                     }
413               }
414         }
```

Workbench.NaturalLanguage.Grammar.Editor.GrammarFeatureEdit.cs

```
////////////////////////////////////////////////////////////////////
/// WORKBENCH                                                    ///
///                                                               ///
///                                                              ///
///                                                              ///
/// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
////////////////////////////////////////////////////////////////////
/// Revision information:                                       ///
///   $Workfile:: GrammarFeatureEdit.cs                       $ ///
///   $Revision:: 1                                           $ ///
///   $Author:: Ron_van_gog, Kamal_Sayah                      $ ///
///   $Date:: 24/02/04                                         $ ///
////////////////////////////////////////////////////////////////////
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using Belastingdienst.Windows;

namespace Workbench.NaturalLanguage.Grammar.Editor
{
   // Deze klasse omschrijft de complete GrammarFeatureEdit-form
   public class GrammarFeatureEdit : System.Windows.Forms.Form
   {
          private ArrayList myFeatureSet;
          private GrammarFeature myFeature;
          private bool newFeature;
          private System.Windows.Forms.Label label1;
          private System.Windows.Forms.Label label2;
          private System.Windows.Forms.CheckBox cbAtomic;
          private System.Windows.Forms.TextBox tbName;
          private System.Windows.Forms.TextBox tbEquationId;
          private System.Windows.Forms.TextBox tbValue;
          private System.Windows.Forms.Label label3;
          private System.Windows.Forms.Button cbCancel;
          private System.Windows.Forms.Button cbOk;
          private System.Windows.Forms.Label lbCaptionNew;
          private System.Windows.Forms.Label lbCaptionEdit;
          /// <summary>
          /// Required designer variable.
          /// </summary>
          private System.ComponentModel.Container components = null;

          public GrammarFeatureEdit()
          {
                 //
                 // Required for Windows Form Designer support
                 //
                 InitializeComponent();

                 //
                 // TODO: Add any constructor code after InitializeComponent call
                 //
          }

          /// <summary>
          /// Clean up any resources being used.
          /// </summary>
          protected override void Dispose( bool disposing )
          {
                 if( disposing )
                 {
                        if(components != null)
                        {
                               components.Dispose();
                        }
                 }
                 base.Dispose( disposing );
          }

          #region Windows Form Designer generated code
          /// <summary>
          /// Required method for Designer support - do not modify
          /// the contents of this method with the code editor.
          /// </summary>
          private void InitializeComponent()
          {
                 this.label1 = new System.Windows.Forms.Label();
                 this.label2 = new System.Windows.Forms.Label();
                 this.cbAtomic = new System.Windows.Forms.CheckBox();
                 this.tbName = new System.Windows.Forms.TextBox();
                 this.tbEquationId = new System.Windows.Forms.TextBox();
                 this.tbValue = new System.Windows.Forms.TextBox();
```

```
            this.label3 = new System.Windows.Forms.Label();
            this.cbCancel = new System.Windows.Forms.Button();
            this.cbOk = new System.Windows.Forms.Button();
            this.lbCaptionNew = new System.Windows.Forms.Label();
            this.lbCaptionEdit = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.Location = new System.Drawing.Point(8, 16);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(48, 16);
            this.label1.TabIndex = 6;
            this.label1.Text = "Name";
            //
            // label2
            //
            this.label2.Location = new System.Drawing.Point(8, 40);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(64, 16);
            this.label2.TabIndex = 7;
            this.label2.Text = "Equation Id";
            //
            // cbAtomic
            //
            this.cbAtomic.CheckAlign = System.Drawing.ContentAlignment.MiddleRight;
            this.cbAtomic.ImageAlign = System.Drawing.ContentAlignment.MiddleLeft;
            this.cbAtomic.Location = new System.Drawing.Point(8, 88);
            this.cbAtomic.Name = "cbAtomic";
            this.cbAtomic.Size = new System.Drawing.Size(80, 24);
            this.cbAtomic.TabIndex = 3;
            this.cbAtomic.Text = "Atomic";
            this.cbAtomic.CheckedChanged += new
            System.EventHandler(this.cbAtomic_CheckedChanged);
            //
            // tbName
            //
            this.tbName.Location = new System.Drawing.Point(72, 8);
            this.tbName.Name = "tbName";
            this.tbName.Size = new System.Drawing.Size(240, 20);
            this.tbName.TabIndex = 0;
            this.tbName.Text = "";
            //
            // tbEquationId
            //
            this.tbEquationId.Location = new System.Drawing.Point(72, 32);
            this.tbEquationId.Name = "tbEquationId";
            this.tbEquationId.Size = new System.Drawing.Size(240, 20);
            this.tbEquationId.TabIndex = 1;
            this.tbEquationId.Text = "";
            //
            // tbValue
            //
            this.tbValue.Location = new System.Drawing.Point(72, 56);
            this.tbValue.Name = "tbValue";
            this.tbValue.Size = new System.Drawing.Size(240, 20);
            this.tbValue.TabIndex = 2;
            this.tbValue.Text = "";
            //
            // label3
            //
            this.label3.Location = new System.Drawing.Point(8, 64);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(56, 16);
            this.label3.TabIndex = 8;
            this.label3.Text = "Value";
            //
            // cbCancel
            //
            this.cbCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.cbCancel.Location = new System.Drawing.Point(168, 120);
            this.cbCancel.Name = "cbCancel";
            this.cbCancel.Size = new System.Drawing.Size(64, 24);
            this.cbCancel.TabIndex = 5;
            this.cbCancel.Text = "Cancel";
            this.cbCancel.Click += new System.EventHandler(this.cbCancel_Click);
            //
            // cbOk
            //
            this.cbOk.Location = new System.Drawing.Point(96, 120);
            this.cbOk.Name = "cbOk";
            this.cbOk.Size = new System.Drawing.Size(64, 24);
            this.cbOk.TabIndex = 4;
            this.cbOk.Text = "Ok";
            this.cbOk.Click += new System.EventHandler(this.cbOk_Click);
            //
            // lbCaptionNew
            //
```

```
173                     this.lbCaptionNew.Location = new System.Drawing.Point(192, 96);
174                     this.lbCaptionNew.Name = "lbCaptionNew";
175                     this.lbCaptionNew.Size = new System.Drawing.Size(120, 23);
176                     this.lbCaptionNew.TabIndex = 10;
177                     this.lbCaptionNew.Text = "New grammar feature";
178                     this.lbCaptionNew.Visible = false;
179                     //
180                     // lbCaptionEdit
181                     //
182                     this.lbCaptionEdit.Location = new System.Drawing.Point(192, 80);
183                     this.lbCaptionEdit.Name = "lbCaptionEdit";
184                     this.lbCaptionEdit.Size = new System.Drawing.Size(120, 23);
185                     this.lbCaptionEdit.TabIndex = 9;
186                     this.lbCaptionEdit.Text = "Edit grammar feature";
187                     this.lbCaptionEdit.Visible = false;
188                     //
189                     // GrammarFeatureEdit
190                     //
191                     this.AcceptButton = this.cbOk;
192                     this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
193                     this.CancelButton = this.cbCancel;
194                     this.ClientSize = new System.Drawing.Size(320, 149);
195                     this.Controls.Add(this.lbCaptionNew);
196                     this.Controls.Add(this.lbCaptionEdit);
197                     this.Controls.Add(this.cbCancel);
198                     this.Controls.Add(this.cbOk);
199                     this.Controls.Add(this.label3);
200                     this.Controls.Add(this.tbValue);
201                     this.Controls.Add(this.tbEquationId);
202                     this.Controls.Add(this.tbName);
203                     this.Controls.Add(this.cbAtomic);
204                     this.Controls.Add(this.label2);
205                     this.Controls.Add(this.label1);
206                     this.Name = "GrammarFeatureEdit";
207                     this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
208                     this.Text = "GrammarFeatureEdit";
209                     this.ResumeLayout(false);
210
211                 }
212                 #endregion
213
214                 private void cbOk_Click(object sender, System.EventArgs e)
215                 {
216                     if (newFeature)
217                     {
218                         myFeature = new GrammarFeature(tbName.Text);
219                         myFeatureSet.Add(myFeature);
220                     }
221                     myFeature.Name = tbName.Text;
222                     myFeature.Atomic = cbAtomic.Checked;
223                     if (myFeature.Atomic)
224                     {
225                         myFeature.FeatureValue = tbValue.Text;
226                     }
227                     else
228                     {
229                         myFeature.FeatureValue = null;
230                     }
231                     if (tbEquationId.Text.Trim().Equals(""))
232                     {
233                         myFeature.EquationId = -1;
234                     }
235                     else
236                     {
237                         try
238                         {
239                             int eqId = Int32.Parse(tbEquationId.Text);
240                             if (eqId > 0)
241                             {
242                                 myFeature.EquationId = eqId;
243                             }
244                             else
245                             {
246                             System.Windows.Forms.MessageBox.Show(this,"EquitionId moet
247                             groter dan 0 zijn!");
248                             }
249                         }
250                         catch(Exception exception)
251                         {
252                         System.Windows.Forms.MessageBox.Show(this,"U heeft geen geldig
253                         EquitionId ingevuld!");
254                         }
255                     }
256                     this.Close();
257                 }
258
259                 private void cbCancel_Click(object sender, System.EventArgs e)
260                 {
```

```
261                         this.Close();
262                   }
263
264                   // Deze methode wordt in de GrammarForm (de hoofdform) aangeroepen om de
265                   //uiteindelijke edit te verwerken in het klassemodel
266                   public void DoEdit(GrammarFeature item)
267                   {
268                         this.Text = lbCaptionEdit.Text;
269                         this.myFeature = item;
270                         this.newFeature = false;
271                         this.tbName.Text = item.Name;
272                         if (item.EquationId.Equals(-1))
273                         {
274                               this.tbEquationId.Text = "";
275                         }
276                         else
277                         {
278                               this.tbEquationId.Text = item.EquationId.ToString();
279                         }
280                         this.tbValue.Text = item.FeatureValue;
281                         this.cbAtomic.Checked = item.Atomic;
282                         if (cbAtomic.Checked)
283                         {
284                               this.tbValue.Enabled = true;
285                         }
286                         else
287                         {
288                               this.tbValue.Enabled = false;
289                         }
290                         if (item.Fixed)
291                         {
292                               this.tbName.Enabled = false;
293                               this.tbValue.Enabled = false;
294                               this.cbAtomic.Enabled = false;
295                         }
296                         this.ShowDialog();
297                   }
298
299                   public void DoNew(ArrayList item)
300                   {
301                         this.Text = lbCaptionEdit.Text;
302                         this.myFeatureSet = item;
303                         this.newFeature = true;
304                         this.tbName.Text = "";
305                         this.tbEquationId.Text = "";
306                         this.tbValue.Text = "";
307                         this.cbAtomic.Checked = false;
308                         this.tbValue.Enabled = false;
309                         this.ShowDialog();
310                   }
311
312             // Een event die zorgt voor de correcte gevolgen van een verandering van de "atomische
313             //waarde"-check.
314                   private void cbAtomic_CheckedChanged(object sender, System.EventArgs e)
315                   {
316                         if (cbAtomic.Checked)
317                         {
318                               this.tbValue.Enabled = true;
319                         }
320                         else
321                         {
322                               this.tbValue.Enabled = false;
323                         }
324                   }
325             }
326       }
327
```

Workbench.NaturalLanguage.Grammar.Editor.GrammarRuleEdit.cs

```
1   ////////////////////////////////////////////////////////////////
2   /// WORKBENCH                                                ///
3   ///                                                          ///
4   ///                                                          ///
5   ///                                                          ///
6   /// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
7   ////////////////////////////////////////////////////////////////
8   /// Revision information:                                    ///
9   ///   $Workfile:: GrammarRuleEdit.cs                       $ ///
10  ///   $Revision:: 1                                        $ ///
11  ///   $Author:: Ron_van_gog, Kamal_Sayah                   $ ///
12  ///   $Date:: 24/02/04                                     $ ///
13  ////////////////////////////////////////////////////////////////
14  using System;
15  using System.Drawing;
16  using System.Collections;
17  using System.ComponentModel;
18  using System.Windows.Forms;
19
20  namespace Workbench.NaturalLanguage.Grammar.Editor
21  {
22      /// <summary>
23      /// Summary description for GrammarRuleEdit.
24      /// </summary>
25
26      // Deze klasse beschrijft de form die gebruitk wordt bij het editten van een GrammarRule
27      public class GrammarRuleEdit : System.Windows.Forms.Form
28      {
29              private GrammarRuleCollection myCollection;
30              private GrammarRule myRule;
31              private bool newRule;
32              private System.Windows.Forms.TextBox tbName;
33              private System.Windows.Forms.Label label1;
34              private System.Windows.Forms.TextBox tbDescription;
35              private System.Windows.Forms.Label label2;
36              private System.Windows.Forms.Button cbCancel;
37              private System.Windows.Forms.Button cbOk;
38              private System.Windows.Forms.Label lbCaptionNew;
39              private System.Windows.Forms.Label lbCaptionEdit;
40              /// <summary>
41              /// Required designer variable.
42              /// </summary>
43              private System.ComponentModel.Container components = null;
44
45              public GrammarRuleEdit()
46              {
47                      //
48                      // Required for Windows Form Designer support
49                      //
50                      InitializeComponent();
51
52                      //
53                      // TODO: Add any constructor code after InitializeComponent call
54                      //
55              }
56
57              /// <summary>
58              /// Clean up any resources being used.
59              /// </summary>
60              protected override void Dispose( bool disposing )
61              {
62                      if( disposing )
63                      {
64                              if(components != null)
65                              {
66                                      components.Dispose();
67                              }
68                      }
69                      base.Dispose( disposing );
70              }
71
72              #region Windows Form Designer generated code
73              /// <summary>
74              /// Required method for Designer support - do not modify
75              /// the contents of this method with the code editor.
76              /// </summary>
77              private void InitializeComponent()
78              {
79                      this.tbName = new System.Windows.Forms.TextBox();
80                      this.label1 = new System.Windows.Forms.Label();
81                      this.tbDescription = new System.Windows.Forms.TextBox();
82                      this.label2 = new System.Windows.Forms.Label();
83                      this.cbCancel = new System.Windows.Forms.Button();
84                      this.cbOk = new System.Windows.Forms.Button();
85                      this.lbCaptionNew = new System.Windows.Forms.Label();
```

```
            this.lbCaptionEdit = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // tbName
            //
            this.tbName.Location = new System.Drawing.Point(80, 8);
            this.tbName.Name = "tbName";
            this.tbName.Size = new System.Drawing.Size(304, 20);
            this.tbName.TabIndex = 0;
            this.tbName.Text = "";
            //
            // label1
            //
            this.label1.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.label1.Location = new System.Drawing.Point(8, 8);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(40, 16);
            this.label1.TabIndex = 4;
            this.label1.Text = "Name";
            //
            // tbDescription
            //
            this.tbDescription.Location = new System.Drawing.Point(80, 32);
            this.tbDescription.Multiline = true;
            this.tbDescription.Name = "tbDescription";
            this.tbDescription.Size = new System.Drawing.Size(304, 168);
            this.tbDescription.TabIndex = 1;
            this.tbDescription.Text = "";
            //
            // label2
            //
            this.label2.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.label2.Location = new System.Drawing.Point(8, 40);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(72, 16);
            this.label2.TabIndex = 5;
            this.label2.Text = "Description";
            //
            // cbCancel
            //
            this.cbCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.cbCancel.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.cbCancel.Location = new System.Drawing.Point(200, 208);
            this.cbCancel.Name = "cbCancel";
            this.cbCancel.Size = new System.Drawing.Size(64, 24);
            this.cbCancel.TabIndex = 3;
            this.cbCancel.Text = "Cancel";
            this.cbCancel.Click += new System.EventHandler(this.cbCancel_Click);
            //
            // cbOk
            //
            this.cbOk.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.cbOk.Location = new System.Drawing.Point(128, 208);
            this.cbOk.Name = "cbOk";
            this.cbOk.Size = new System.Drawing.Size(64, 24);
            this.cbOk.TabIndex = 2;
            this.cbOk.Text = "Ok";
            this.cbOk.Click += new System.EventHandler(this.cbOk_Click);
            //
            // lbCaptionNew
            //
            this.lbCaptionNew.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.lbCaptionNew.Location = new System.Drawing.Point(8, 208);
            this.lbCaptionNew.Name = "lbCaptionNew";
            this.lbCaptionNew.Size = new System.Drawing.Size(120, 23);
            this.lbCaptionNew.TabIndex = 7;
            this.lbCaptionNew.Text = "New grammar rule";
            this.lbCaptionNew.Visible = false;
            //
            // lbCaptionEdit
            //
            this.lbCaptionEdit.ImeMode = System.Windows.Forms.ImeMode.NoControl;
            this.lbCaptionEdit.Location = new System.Drawing.Point(8, 184);
            this.lbCaptionEdit.Name = "lbCaptionEdit";
            this.lbCaptionEdit.Size = new System.Drawing.Size(120, 23);
            this.lbCaptionEdit.TabIndex = 6;
            this.lbCaptionEdit.Text = "Edit grammar rule";
            this.lbCaptionEdit.Visible = false;
            //
            // GrammarRuleEdit
            //
            this.AcceptButton = this.cbOk;
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.CancelButton = this.cbCancel;
            this.ClientSize = new System.Drawing.Size(392, 237);
            this.Controls.Add(this.lbCaptionNew);
            this.Controls.Add(this.lbCaptionEdit);
            this.Controls.Add(this.cbCancel);
```

```
174                         this.Controls.Add(this.cbOk);
175                         this.Controls.Add(this.tbDescription);
176                         this.Controls.Add(this.label2);
177                         this.Controls.Add(this.tbName);
178                         this.Controls.Add(this.label1);
179                         this.Name = "GrammarRuleEdit";
180                         this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
181                         this.Text = "GrammarRuleEdit";
182                         this.ResumeLayout(false);
183
184                   }
185                   #endregion
186
187                   // Afhandeling van de OK-button
188                   private void cbOk_Click(object sender, System.EventArgs e)
189                   {
190                         if (newRule)
191                         {
192                               myRule = new GrammarRule(tbName.Text);
193                               myRule.isNew = true;
194                               myCollection.Add(myRule);
195                         }
196                         myRule.Name = tbName.Text;
197                         myRule.Description = tbDescription.Text;
198                         this.Close();
199                   }
200
201                   private void cbCancel_Click(object sender, System.EventArgs e)
202                   {
203                         this.Close();
204                   }
205
206                   // Het verwerken van een edit
207                   public void DoEdit(GrammarRule item)
208                   {
209                         this.Text = lbCaptionEdit.Text;
210                         this.myRule = item;
211                         this.newRule = false;
212                         this.tbName.Text = item.Name;
213                         this.tbDescription.Text = item.Description;
214                         this.ShowDialog();
215                   }
216
217                   // Het verwerken van een nieuwe-regel-toevoeging
218                   public void DoNew(GrammarRuleCollection item)
219                   {
220                         this.Text = lbCaptionEdit.Text;
221                         this.myCollection = item;
222                         this.newRule = true;
223                         this.tbName.Text = "";
224                         this.tbDescription.Text = "";
225                         this.ShowDialog();
226                   }
227
228             }
229       }
```

Workbench.NaturalLanguage.Grammar.Editor.GrammarForm.cs

```
//////////////////////////////////////////////////////////////////
/// WORKBENCH                                                   ///
///                                                             ///
///                                                             ///
///                                                             ///
/// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
//////////////////////////////////////////////////////////////////
/// Revision information:                                       ///
///   $Workfile:: GrammarForm.cs                              $ ///
///   $Revision:: 1                                            $ ///
///   $Author:: Ron_van_gog, Kamal_Sayah                      $ ///
///   $Date:: 24/02/04                                        $ ///
//////////////////////////////////////////////////////////////////

using System;
using System.Xml;
using System.Data;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using Belastingdienst.Windows;

namespace Workbench.NaturalLanguage.Grammar.Editor
{
   /// <summary>
   /// Summary description for GrammarForm.
   /// </summary>

   // Deze klasse omschrijft het complete hoofdscherm van de Grammartool
   public class GrammarForm : System.Windows.Forms.Form
   {
          private System.Windows.Forms.TreeNode myNode;
          private GrammarTreeModel myGrammarModel;
          private System.Windows.Forms.ContextMenu contextMenu1;
          private Belastingdienst.Windows.Forms.TreeModelView grammarTreeView;
          private System.Windows.Forms.MenuItem menuEdit;
          private System.Windows.Forms.MenuItem menuNewElement;
          private System.Windows.Forms.MenuItem menuNewFeature;
          private System.Windows.Forms.MenuItem menuDelete;
          private System.Windows.Forms.MenuItem menuMoveUp;
          private System.Windows.Forms.MenuItem menuMoveDown;
          private System.Windows.Forms.MenuItem menuNewRule;
          private System.Windows.Forms.MenuItem menuItem2;
          private System.Windows.Forms.MenuItem menuItem3;
          private System.Windows.Forms.Panel panel1;
          private System.Windows.Forms.Button cbSave;
          private System.Windows.Forms.Button cbCancel;
          private Belastingdienst.Windows.Forms.MessageBox DeleteMessageBox;
          private System.Windows.Forms.Button cbPrint;
          private System.Windows.Forms.SaveFileDialog PrintTo;
          private Belastingdienst.Windows.Forms.MessageBox CancelMessageBox;
          /// <summary>
          /// Required designer variable.
          /// </summary>
          private System.ComponentModel.Container components = null;

          public GrammarForm()
          {
                 //
                 // Required for Windows Form Designer support
                 //
                 InitializeComponent();

                 //
                 // TODO: Add any constructor code after InitializeComponent call
                 //
          }

          /// <summary>
          /// Clean up any resources being used.
          /// </summary>
          protected override void Dispose( bool disposing )
          {
                 if( disposing )
                 {
                        if(components != null)
                        {
                               components.Dispose();
                        }
                 }
                 base.Dispose( disposing );
          }

          #region Windows Form Designer generated code
```

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        System.Resources.ResourceManager resources = new
        System.Resources.ResourceManager(typeof(GrammarForm));
        this.myGrammarModel = new
        Workbench.NaturalLanguage.Grammar.Editor.GrammarTreeModel();
        this.contextMenu1 = new System.Windows.Forms.ContextMenu();
        this.menuEdit = new System.Windows.Forms.MenuItem();
        this.menuDelete = new System.Windows.Forms.MenuItem();
        this.menuItem3 = new System.Windows.Forms.MenuItem();
        this.menuNewRule = new System.Windows.Forms.MenuItem();
        this.menuNewElement = new System.Windows.Forms.MenuItem();
        this.menuNewFeature = new System.Windows.Forms.MenuItem();
        this.menuItem2 = new System.Windows.Forms.MenuItem();
        this.menuMoveUp = new System.Windows.Forms.MenuItem();
        this.menuMoveDown = new System.Windows.Forms.MenuItem();
        this.grammarTreeView = new Belastingdienst.Windows.Forms.TreeModelView();
        this.panel1 = new System.Windows.Forms.Panel();
        this.cbPrint = new System.Windows.Forms.Button();
        this.cbCancel = new System.Windows.Forms.Button();
        this.cbSave = new System.Windows.Forms.Button();
        this.DeleteMessageBox = new Belastingdienst.Windows.Forms.MessageBox();
        this.PrintTo = new System.Windows.Forms.SaveFileDialog();
        this.CancelMessageBox = new Belastingdienst.Windows.Forms.MessageBox();
        this.panel1.SuspendLayout();
        this.SuspendLayout();
        //
        // contextMenu1
        //
        this.contextMenu1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
        this.menuEdit,
        this.menuDelete,
        this.menuItem3,
        this.menuNewRule,
        this.menuNewElement,
        this.menuNewFeature,
        this.menuItem2,
        this.menuMoveUp,
        this.menuMoveDown});
        this.contextMenu1.RightToLeft =
        ((System.Windows.Forms.RightToLeft)(resources.GetObject("contextMenu1.RightToLeft")))
        ;
        this.contextMenu1.Popup += new System.EventHandler(this.contextMenu1_Popup);
        //
        // menuEdit
        //
        this.menuEdit.Enabled = ((bool)(resources.GetObject("menuEdit.Enabled")));
        this.menuEdit.Index = 0;
        this.menuEdit.Shortcut =
        ((System.Windows.Forms.Shortcut)(resources.GetObject("menuEdit.Shortcut")));
        this.menuEdit.ShowShortcut = ((bool)(resources.GetObject("menuEdit.ShowShortcut")));
        this.menuEdit.Text = resources.GetString("menuEdit.Text");
        this.menuEdit.Visible = ((bool)(resources.GetObject("menuEdit.Visible")));
        this.menuEdit.Click += new System.EventHandler(this.menuEdit_Click);
        //
        // menuDelete
        //
        this.menuDelete.Enabled = ((bool)(resources.GetObject("menuDelete.Enabled")));
        this.menuDelete.Index = 1;
        this.menuDelete.Shortcut =
        ((System.Windows.Forms.Shortcut)(resources.GetObject("menuDelete.Shortcut")));
        this.menuDelete.ShowShortcut =
        ((bool)(resources.GetObject("menuDelete.ShowShortcut")));
        this.menuDelete.Text = resources.GetString("menuDelete.Text");
        this.menuDelete.Visible = ((bool)(resources.GetObject("menuDelete.Visible")));
        this.menuDelete.Click += new System.EventHandler(this.menuDelete_Click);
        //
        // menuItem3
        //
        this.menuItem3.Enabled = ((bool)(resources.GetObject("menuItem3.Enabled")));
        this.menuItem3.Index = 2;
        this.menuItem3.Shortcut =
        ((System.Windows.Forms.Shortcut)(resources.GetObject("menuItem3.Shortcut")));
        this.menuItem3.ShowShortcut =
        ((bool)(resources.GetObject("menuItem3.ShowShortcut")));
        this.menuItem3.Text = resources.GetString("menuItem3.Text");
        this.menuItem3.Visible = ((bool)(resources.GetObject("menuItem3.Visible")));
        //
        // menuNewRule
        //
        this.menuNewRule.Enabled = ((bool)(resources.GetObject("menuNewRule.Enabled")));
        this.menuNewRule.Index = 3;
        this.menuNewRule.Shortcut =
        ((System.Windows.Forms.Shortcut)(resources.GetObject("menuNewRule.Shortcut")));
```

```
174                     this.menuNewRule.ShowShortcut =
175                     ((bool)(resources.GetObject("menuNewRule.ShowShortcut")));
176                     this.menuNewRule.Text = resources.GetString("menuNewRule.Text");
177                     this.menuNewRule.Visible = ((bool)(resources.GetObject("menuNewRule.Visible")));
178                     this.menuNewRule.Click += new System.EventHandler(this.menuNewRule_Click);
179                     //
180                     // menuNewElement
181                     //
182                     this.menuNewElement.Enabled =
183                     ((bool)(resources.GetObject("menuNewElement.Enabled")));
184                     this.menuNewElement.Index = 4;
185                     this.menuNewElement.Shortcut =
186                     ((System.Windows.Forms.Shortcut)(resources.GetObject("menuNewElement.Shortcut")));
187                     this.menuNewElement.ShowShortcut =
188                     ((bool)(resources.GetObject("menuNewElement.ShowShortcut")));
189                     this.menuNewElement.Text = resources.GetString("menuNewElement.Text");
190                     this.menuNewElement.Visible =
191                     ((bool)(resources.GetObject("menuNewElement.Visible")));
192                     this.menuNewElement.Click += new System.EventHandler(this.menuNewElement_Click);
193                     //
194                     // menuNewFeature
195                     //
196                     this.menuNewFeature.Enabled =
197                     ((bool)(resources.GetObject("menuNewFeature.Enabled")));
198                     this.menuNewFeature.Index = 5;
199                     this.menuNewFeature.Shortcut =
200                     ((System.Windows.Forms.Shortcut)(resources.GetObject("menuNewFeature.Shortcut")));
201                     this.menuNewFeature.ShowShortcut =
202                     ((bool)(resources.GetObject("menuNewFeature.ShowShortcut")));
203                     this.menuNewFeature.Text = resources.GetString("menuNewFeature.Text");
204                     this.menuNewFeature.Visible =
205                     ((bool)(resources.GetObject("menuNewFeature.Visible")));
206                     this.menuNewFeature.Click += new System.EventHandler(this.menuNewFeature_Click);
207                     //
208                     // menuItem2
209                     //
210                     this.menuItem2.Enabled = ((bool)(resources.GetObject("menuItem2.Enabled")));
211                     this.menuItem2.Index = 6;
212                     this.menuItem2.Shortcut =
213                     ((System.Windows.Forms.Shortcut)(resources.GetObject("menuItem2.Shortcut")));
214                     this.menuItem2.ShowShortcut =
215                     ((bool)(resources.GetObject("menuItem2.ShowShortcut")));
216                     this.menuItem2.Text = resources.GetString("menuItem2.Text");
217                     this.menuItem2.Visible = ((bool)(resources.GetObject("menuItem2.Visible")));
218                     //
219                     // menuMoveUp
220                     //
221                     this.menuMoveUp.Enabled = ((bool)(resources.GetObject("menuMoveUp.Enabled")));
222                     this.menuMoveUp.Index = 7;
223                     this.menuMoveUp.Shortcut =
224                     ((System.Windows.Forms.Shortcut)(resources.GetObject("menuMoveUp.Shortcut")));
225                     this.menuMoveUp.ShowShortcut =
226                     ((bool)(resources.GetObject("menuMoveUp.ShowShortcut")));
227                     this.menuMoveUp.Text = resources.GetString("menuMoveUp.Text");
228                     this.menuMoveUp.Visible = ((bool)(resources.GetObject("menuMoveUp.Visible")));
229                     this.menuMoveUp.Click += new System.EventHandler(this.menuMoveUp_Click);
230                     //
231                     // menuMoveDown
232                     //
233                     this.menuMoveDown.Enabled = ((bool)(resources.GetObject("menuMoveDown.Enabled")));
234                     this.menuMoveDown.Index = 8;
235                     this.menuMoveDown.Shortcut =
236                     ((System.Windows.Forms.Shortcut)(resources.GetObject("menuMoveDown.Shortcut")));
237                     this.menuMoveDown.ShowShortcut =
238                     ((bool)(resources.GetObject("menuMoveDown.ShowShortcut")));
239                     this.menuMoveDown.Text = resources.GetString("menuMoveDown.Text");
240                     this.menuMoveDown.Visible = ((bool)(resources.GetObject("menuMoveDown.Visible")));
241                     this.menuMoveDown.Click += new System.EventHandler(this.menuMoveDown_Click);
242                     //
243                     // grammarTreeView
244                     //
245                     this.grammarTreeView.AccessibleDescription =
246                     resources.GetString("grammarTreeView.AccessibleDescription");
247                     this.grammarTreeView.AccessibleName =
248                     resources.GetString("grammarTreeView.AccessibleName");
249                     this.grammarTreeView.Anchor =
250                     ((System.Windows.Forms.AnchorStyles)(resources.GetObject("grammarTreeView.Anchor")));
251                     this.grammarTreeView.BackgroundImage =
252                     ((System.Drawing.Image)(resources.GetObject("grammarTreeView.BackgroundImage")));
253                     this.grammarTreeView.ContextMenu = this.contextMenu1;
254                     this.grammarTreeView.Dock =
255                     ((System.Windows.Forms.DockStyle)(resources.GetObject("grammarTreeView.Dock")));
256                     this.grammarTreeView.Enabled =
257                     ((bool)(resources.GetObject("grammarTreeView.Enabled")));
258                     this.grammarTreeView.Font =
259                     ((System.Drawing.Font)(resources.GetObject("grammarTreeView.Font")));
260                     this.grammarTreeView.ImageIndex =
261                     ((int)(resources.GetObject("grammarTreeView.ImageIndex")));
```

```
            this.grammarTreeView.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("grammarTreeView.ImeMode")));
            this.grammarTreeView.Indent = ((int)(resources.GetObject("grammarTreeView.Indent")));
            this.grammarTreeView.ItemHeight =
            ((int)(resources.GetObject("grammarTreeView.ItemHeight")));
            this.grammarTreeView.Location =
            ((System.Drawing.Point)(resources.GetObject("grammarTreeView.Location")));
            this.grammarTreeView.Model = this.myGrammarModel;
            this.grammarTreeView.Name = "grammarTreeView";
            this.grammarTreeView.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("grammarTreeView.RightToLeft"
            )));
            this.grammarTreeView.SelectedImageIndex =
            ((int)(resources.GetObject("grammarTreeView.SelectedImageIndex")));
            this.grammarTreeView.ShowRoot = false;
            this.grammarTreeView.Size =
            ((System.Drawing.Size)(resources.GetObject("grammarTreeView.Size")));
            this.grammarTreeView.TabIndex =
            ((int)(resources.GetObject("grammarTreeView.TabIndex")));
            this.grammarTreeView.Text = resources.GetString("grammarTreeView.Text");
            this.grammarTreeView.Visible =
            ((bool)(resources.GetObject("grammarTreeView.Visible")));
            this.grammarTreeView.MouseDown += new
            System.Windows.Forms.MouseEventHandler(this.grammarTreeView_MouseDown);
            //
            // panel1
            //
            this.panel1.AccessibleDescription =
            resources.GetString("panel1.AccessibleDescription");
            this.panel1.AccessibleName = resources.GetString("panel1.AccessibleName");
            this.panel1.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("panel1.Anchor")));
            this.panel1.AutoScroll = ((bool)(resources.GetObject("panel1.AutoScroll")));
            this.panel1.AutoScrollMargin =
            ((System.Drawing.Size)(resources.GetObject("panel1.AutoScrollMargin")));
            this.panel1.AutoScrollMinSize =
            ((System.Drawing.Size)(resources.GetObject("panel1.AutoScrollMinSize")));
            this.panel1.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("panel1.BackgroundImage")));
            this.panel1.Controls.Add(this.cbPrint);
            this.panel1.Controls.Add(this.cbCancel);
            this.panel1.Controls.Add(this.cbSave);
            this.panel1.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("panel1.Dock")));
            this.panel1.Enabled = ((bool)(resources.GetObject("panel1.Enabled")));
            this.panel1.Font = ((System.Drawing.Font)(resources.GetObject("panel1.Font")));
            this.panel1.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("panel1.ImeMode")));
            this.panel1.Location =
            ((System.Drawing.Point)(resources.GetObject("panel1.Location")));
            this.panel1.Name = "panel1";
            this.panel1.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("panel1.RightToLeft")));
            this.panel1.Size = ((System.Drawing.Size)(resources.GetObject("panel1.Size")));
            this.panel1.TabIndex = ((int)(resources.GetObject("panel1.TabIndex")));
            this.panel1.Text = resources.GetString("panel1.Text");
            this.panel1.Visible = ((bool)(resources.GetObject("panel1.Visible")));
            //
            // cbPrint
            //
            this.cbPrint.AccessibleDescription =
            resources.GetString("cbPrint.AccessibleDescription");
            this.cbPrint.AccessibleName = resources.GetString("cbPrint.AccessibleName");
            this.cbPrint.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbPrint.Anchor")));
            this.cbPrint.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("cbPrint.BackgroundImage")));
            this.cbPrint.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("cbPrint.Dock")));
            this.cbPrint.Enabled = ((bool)(resources.GetObject("cbPrint.Enabled")));
            this.cbPrint.FlatStyle =
            ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbPrint.FlatStyle")));
            this.cbPrint.Font = ((System.Drawing.Font)(resources.GetObject("cbPrint.Font")));
            this.cbPrint.Image = ((System.Drawing.Image)(resources.GetObject("cbPrint.Image")));
            this.cbPrint.ImageAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbPrint.ImageAlign")));
            this.cbPrint.ImageIndex = ((int)(resources.GetObject("cbPrint.ImageIndex")));
            this.cbPrint.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("cbPrint.ImeMode")));
            this.cbPrint.Location =
            ((System.Drawing.Point)(resources.GetObject("cbPrint.Location")));
            this.cbPrint.Name = "cbPrint";
            this.cbPrint.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbPrint.RightToLeft")));
            this.cbPrint.Size = ((System.Drawing.Size)(resources.GetObject("cbPrint.Size")));
            this.cbPrint.TabIndex = ((int)(resources.GetObject("cbPrint.TabIndex")));
            this.cbPrint.Text = resources.GetString("cbPrint.Text");
```

```
            this.cbPrint.TextAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbPrint.TextAlign")));
            this.cbPrint.Visible = ((bool)(resources.GetObject("cbPrint.Visible")));
            this.cbPrint.Click += new System.EventHandler(this.cbPrint_Click);
            //
            // cbCancel
            //
            this.cbCancel.AccessibleDescription =
            resources.GetString("cbCancel.AccessibleDescription");
            this.cbCancel.AccessibleName = resources.GetString("cbCancel.AccessibleName");
            this.cbCancel.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbCancel.Anchor")));
            this.cbCancel.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("cbCancel.BackgroundImage")));
            this.cbCancel.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.cbCancel.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("cbCancel.Dock")));
            this.cbCancel.Enabled = ((bool)(resources.GetObject("cbCancel.Enabled")));
            this.cbCancel.FlatStyle =
            ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbCancel.FlatStyle")));
            this.cbCancel.Font = ((System.Drawing.Font)(resources.GetObject("cbCancel.Font")));
            this.cbCancel.Image =
            ((System.Drawing.Image)(resources.GetObject("cbCancel.Image")));
            this.cbCancel.ImageAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbCancel.ImageAlign")));
            this.cbCancel.ImageIndex = ((int)(resources.GetObject("cbCancel.ImageIndex")));
            this.cbCancel.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("cbCancel.ImeMode")));
            this.cbCancel.Location =
            ((System.Drawing.Point)(resources.GetObject("cbCancel.Location")));
            this.cbCancel.Name = "cbCancel";
            this.cbCancel.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbCancel.RightToLeft")));
            this.cbCancel.Size = ((System.Drawing.Size)(resources.GetObject("cbCancel.Size")));
            this.cbCancel.TabIndex = ((int)(resources.GetObject("cbCancel.TabIndex")));
            this.cbCancel.Text = resources.GetString("cbCancel.Text");
            this.cbCancel.TextAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbCancel.TextAlign")));
            this.cbCancel.Visible = ((bool)(resources.GetObject("cbCancel.Visible")));
            this.cbCancel.Click += new System.EventHandler(this.cbCancel_Click);
            //
            // cbSave
            //
            this.cbSave.AccessibleDescription =
            resources.GetString("cbSave.AccessibleDescription");
            this.cbSave.AccessibleName = resources.GetString("cbSave.AccessibleName");
            this.cbSave.Anchor =
            ((System.Windows.Forms.AnchorStyles)(resources.GetObject("cbSave.Anchor")));
            this.cbSave.BackgroundImage =
            ((System.Drawing.Image)(resources.GetObject("cbSave.BackgroundImage")));
            this.cbSave.Dock =
            ((System.Windows.Forms.DockStyle)(resources.GetObject("cbSave.Dock")));
            this.cbSave.Enabled = ((bool)(resources.GetObject("cbSave.Enabled")));
            this.cbSave.FlatStyle =
            ((System.Windows.Forms.FlatStyle)(resources.GetObject("cbSave.FlatStyle")));
            this.cbSave.Font = ((System.Drawing.Font)(resources.GetObject("cbSave.Font")));
            this.cbSave.Image = ((System.Drawing.Image)(resources.GetObject("cbSave.Image")));
            this.cbSave.ImageAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbSave.ImageAlign")));
            this.cbSave.ImageIndex = ((int)(resources.GetObject("cbSave.ImageIndex")));
            this.cbSave.ImeMode =
            ((System.Windows.Forms.ImeMode)(resources.GetObject("cbSave.ImeMode")));
            this.cbSave.Location =
            ((System.Drawing.Point)(resources.GetObject("cbSave.Location")));
            this.cbSave.Name = "cbSave";
            this.cbSave.RightToLeft =
            ((System.Windows.Forms.RightToLeft)(resources.GetObject("cbSave.RightToLeft")));
            this.cbSave.Size = ((System.Drawing.Size)(resources.GetObject("cbSave.Size")));
            this.cbSave.TabIndex = ((int)(resources.GetObject("cbSave.TabIndex")));
            this.cbSave.Text = resources.GetString("cbSave.Text");
            this.cbSave.TextAlign =
            ((System.Drawing.ContentAlignment)(resources.GetObject("cbSave.TextAlign")));
            this.cbSave.Visible = ((bool)(resources.GetObject("cbSave.Visible")));
            this.cbSave.Click += new System.EventHandler(this.cbSave_Click);
            //
            // DeleteMessageBox
            //
            this.DeleteMessageBox.Buttons = System.Windows.Forms.MessageBoxButtons.OKCancel;
            this.DeleteMessageBox.Text = resources.GetString("DeleteMessageBox.Text");
            this.DeleteMessageBox.Title = resources.GetString("DeleteMessageBox.Title");
            //
            // PrintTo
            //
            this.PrintTo.DefaultExt = "htm";
            this.PrintTo.FileName = "Grammar.htm";
            this.PrintTo.Filter = resources.GetString("PrintTo.Filter");
            this.PrintTo.Title = resources.GetString("PrintTo.Title");
            //
```

```
437                     // CancelMessageBox
438                     //
439                     this.CancelMessageBox.Buttons = System.Windows.Forms.MessageBoxButtons.OKCancel;
440                     this.CancelMessageBox.Text = resources.GetString("CancelMessageBox.Text");
441                     this.CancelMessageBox.Title = resources.GetString("CancelMessageBox.Title");
442                     //
443                     // GrammarForm
444                     //
445                     this.AccessibleDescription = resources.GetString("$this.AccessibleDescription");
446                     this.AccessibleName = resources.GetString("$this.AccessibleName");
447                     this.AutoScaleBaseSize =
448                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScaleBaseSize")));
449                     this.AutoScroll = ((bool)(resources.GetObject("$this.AutoScroll")));
450                     this.AutoScrollMargin =
451                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScrollMargin")));
452                     this.AutoScrollMinSize =
453                     ((System.Drawing.Size)(resources.GetObject("$this.AutoScrollMinSize")));
454                     this.BackgroundImage =
455                     ((System.Drawing.Image)(resources.GetObject("$this.BackgroundImage")));
456                     this.CancelButton = this.cbCancel;
457                     this.ClientSize = ((System.Drawing.Size)(resources.GetObject("$this.ClientSize")));
458                     this.Controls.Add(this.grammarTreeView);
459                     this.Controls.Add(this.panel1);
460                     this.Enabled = ((bool)(resources.GetObject("$this.Enabled")));
461                     this.Font = ((System.Drawing.Font)(resources.GetObject("$this.Font")));
462                     this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
463                     this.ImeMode =
464                     ((System.Windows.Forms.ImeMode)(resources.GetObject("$this.ImeMode")));
465                     this.Location = ((System.Drawing.Point)(resources.GetObject("$this.Location")));
466                     this.MaximumSize = ((System.Drawing.Size)(resources.GetObject("$this.MaximumSize")));
467                     this.MinimumSize = ((System.Drawing.Size)(resources.GetObject("$this.MinimumSize")));
468                     this.Name = "GrammarForm";
469                     this.RightToLeft =
470                     ((System.Windows.Forms.RightToLeft)(resources.GetObject("$this.RightToLeft")));
471                     this.StartPosition =
472                     ((System.Windows.Forms.FormStartPosition)(resources.GetObject("$this.StartPosition"))
473                     );
474                     this.Text = resources.GetString("$this.Text");
475                     this.Load += new System.EventHandler(this.GrammarForm_Load);
476                     this.panel1.ResumeLayout(false);
477                     this.ResumeLayout(false);
478
479                     }
480                     #endregion
481
482             // In deze methode worden alle productieregels zoals ze op dat moment in de database
483             //aanwezig zijn
484             // ingelezen en gekoppeld aan het interne klassemodel en getoond in de Grammarform (in een
485             //tree structure)
486                     private void GrammarForm_Load(object sender, System.EventArgs e)
487                     {
488                     string queryString = "SELECT * FROM ProductionRules";
489                     string mode = "Translate";
490                     System.Globalization.CultureInfo culture = new
491                     System.Globalization.CultureInfo("nl");
492                     DataTable ProductionRulesTable =
493                     Workbench.NaturalLanguage.Data.NLPDataAccessHelper.GetDataSet(queryString,mode,cultur
494                     e).Tables[0];
495
496                     // Aanmaken van de nieuwe productieregelcollectie
497                     GrammarRuleCollection ruleCollection = new
498                     GrammarRuleCollection(ProductionRulesTable);
499                     this.myGrammarModel.SetRoot(ruleCollection);
500                     this.grammarTreeView.Reset();
501                     }
502
503             private void grammarTreeView_MouseDown(object sender, System.Windows.Forms.MouseEventArgs
504             e)
505                     {
506                             this.myNode = this.grammarTreeView.GetNodeAt(e.X, e.Y);
507                     }
508
509                     private void menuEdit_Click(object sender, System.EventArgs e)
510                     {
511                             object item = this.grammarTreeView.NodeItem(this.myNode);
512
513                     // Als men zich bevindt op een GrammarRule dan wordt de edit (form) van de
514                     //GrammarRule aangeroepen
515                             if (item is GrammarRule)
516                             {
517                                     new GrammarRuleEdit().DoEdit((GrammarRule)item);
518                                     this.grammarTreeView.Reset(this.myNode);
519                             }
520
521                     // Als men zich bevindt op een GrammarElement dan wordt de edit (form) van de
522                     //GrammarElement aangeroepen
523                             else if (item is GrammarElement)
524                             {
```

```
                                new GrammarElementEdit().DoEdit((GrammarElement)item);
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }

                // Als men zich bevindt op een GrammarFeature dan wordt de edit (form) van de
                //GrammarFeature aangeroepen
                        else if (item is GrammarFeature)
                        {
                                new GrammarFeatureEdit().DoEdit((GrammarFeature)item);
                                this.grammarTreeView.Reset(this.myNode);
                        }
                }

                // Het opbouwen van het contextmenu.
                private void contextMenu1_Popup(object sender, System.EventArgs e)
                {
                        object item = this.grammarTreeView.NodeItem(this.myNode);

                        foreach(MenuItem menuItem in this.contextMenu1.MenuItems)
                        {
                                menuItem.Enabled = false;
                        }
                        this.menuNewRule.Enabled = true;
                        if (item is GrammarRule)
                        {
                                this.menuEdit.Enabled = true;
                                this.menuDelete.Enabled = true;
                                this.menuNewElement.Enabled = true;
                        }
                        else if (item is GrammarElement)
                        {
                                this.menuEdit.Enabled = true;
                                this.menuNewFeature.Enabled = true;
                                if (!((GrammarElement)item).Lhs)
                                {
                                        this.menuDelete.Enabled = true;
                                        if (this.myNode.Index>1)
                                        {
                                                this.menuMoveUp.Enabled = true;
                                        }
                                if (this.myNode.Index>0 &&
                                this.myNode.Index<this.myNode.Parent.Nodes.Count-1)
                                        {
                                                this.menuMoveDown.Enabled = true;
                                        }
                                }
                        }
                        else if (item is GrammarFeature)
                        {
                                this.menuEdit.Enabled = true;
                                if (!((GrammarFeature)item).Atomic)
                                {
                                        this.menuNewFeature.Enabled = true;
                                }
                        if (!((GrammarFeature)item).Fixed) // TODO: Eigenlijk moet ook gekeken worden
                        // naar alle subelementen in de boom.
                                {
                                        this.menuDelete.Enabled = true;
                                }
                                if (this.myNode.Index>1)
                                {
                                        this.menuMoveUp.Enabled = true;
                                }
                                if (this.myNode.Index>0 &&
        this.myNode.Index<this.myNode.Parent.Nodes.Count-1)
                                {
                                        this.menuMoveDown.Enabled = true;
                                }
                        }
                }

                // Event voor de afhandeling van de GrammarRule-creatie
                private void menuNewElement_Click(object sender, System.EventArgs e)
                {
                        object item = this.grammarTreeView.NodeItem(this.myNode);
                        if (item is GrammarRule)
                        {
                                new GrammarElementEdit().DoNew((GrammarRule)item);
                                this.grammarTreeView.Reset(this.myNode);
                        }
                }

                // Event voor de afhandeling van de GrammarFeature/Element-creatie
                private void menuNewFeature_Click(object sender, System.EventArgs e)
                {
                        object item = this.grammarTreeView.NodeItem(this.myNode);
                        if (item is GrammarElement)
                        {
```

```
                        new GrammarFeatureEdit().DoNew(((GrammarElement)item).FeatureSet);
                        this.grammarTreeView.Reset(this.myNode);
                }
                else if (item is GrammarFeature)
                {
                        new GrammarFeatureEdit().DoNew(((GrammarFeature)item).FeatureSet);
                        this.grammarTreeView.Reset(this.myNode);
                }
        }

        // Event voor de afhandeling een delete
        private void menuDelete_Click(object sender, System.EventArgs e)
        {
                object item = this.grammarTreeView.NodeItem(this.myNode);
                object parent = this.grammarTreeView.NodeItem(this.myNode.Parent);
                string oldText = DeleteMessageBox.Text;
                string itemName = "";
                if (item is GrammarRule)
                {
                        itemName = ((GrammarRule)item).Name;
                }
                else if (item is GrammarElement)
                {
                        itemName = ((GrammarElement)item).Name;
                }
                else if (item is GrammarFeature)
                {
                        itemName = ((GrammarFeature)item).Name;
                }
                DeleteMessageBox.Text = oldText.Replace("%",itemName);
                DialogResult x = DeleteMessageBox.Show();
                DeleteMessageBox.Text = oldText;
                if (x == DialogResult.OK)
                {
                        if (item is GrammarRule)
                        {
                                if (((GrammarRule)item).isNew)
                                {
                                        ((GrammarRuleCollection)this.grammarTreeView.Model.Root)
                                .RuleCollection.Remove(item);
                                }
                                else
                                {
                                        ((GrammarRule)item).isDeleted = true;
                                }
                                this.grammarTreeView.Reset();
                        }
                        else if (item is GrammarElement)
                        {
                                ((GrammarRule)parent).Rhs.Remove(item);
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }
                        else if (item is GrammarFeature)
                        {
                                if (parent is GrammarElement)
                                {
                                        ((GrammarElement)parent).FeatureSet.Remove(item);
                                }
                                else
                                {
                                        ((GrammarFeature)parent).FeatureSet.Remove(item);
                                }
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }
                }
        }

        // Het verplaatsen van een element (naar boven)
        private void menuMoveUp_Click(object sender, System.EventArgs e)
        {
                object item = this.grammarTreeView.NodeItem(this.myNode);
                if (item is GrammarElement)
                {
                GrammarRule rule =
                (GrammarRule)this.grammarTreeView.NodeItem(this.myNode.Parent);
                        int pos = rule.Rhs.IndexOf(item);
                        if (pos > 0)
                        {
                                rule.Rhs[pos]=rule.Rhs[pos-1];
                                rule.Rhs[pos-1] = item;
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }
                }
                if (item is GrammarFeature)
                {
                GrammarElement element =
                (GrammarElement)this.grammarTreeView.NodeItem(this.myNode.Parent);
                        int pos = element.FeatureSet.IndexOf(item);
```

```
                                element.FeatureSet[pos]= element.FeatureSet[pos-1];
                                element.FeatureSet[pos-1] = item;
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }
                }

                // Het verplaatsen van een element (naar beneden)
                private void menuMoveDown_Click(object sender, System.EventArgs e)
                {
                        object item = this.grammarTreeView.NodeItem(this.myNode);
                        if (item is GrammarElement)
                        {
                        GrammarRule rule =
                        (GrammarRule)this.grammarTreeView.NodeItem(this.myNode.Parent);
                                int pos = rule.Rhs.IndexOf(item);
                                if (pos < rule.Rhs.Count-1)
                                {
                                        rule.Rhs[pos]=rule.Rhs[pos+1];
                                        rule.Rhs[pos+1] = item;
                                        this.grammarTreeView.Reset(this.myNode.Parent);
                                }
                        }
                        if (item is GrammarFeature)
                        {
                        GrammarElement element =
                        (GrammarElement)this.grammarTreeView.NodeItem(this.myNode.Parent);
                                int pos = element.FeatureSet.IndexOf(item);
                                element.FeatureSet[pos]=element.FeatureSet[pos+1];
                                element.FeatureSet[pos+1] = item;
                                this.grammarTreeView.Reset(this.myNode.Parent);
                        }
                }

                // Het aanmaken van een nieuwe regel
                private void menuNewRule_Click(object sender, System.EventArgs e)
                {
                        object item = this.grammarTreeView.Model.Root;
                        new GrammarRuleEdit().DoNew((GrammarRuleCollection)item);
                        this.grammarTreeView.Reset();
                }

                private void cbCancel_Click(object sender, System.EventArgs e)
                {
                        DialogResult x = CancelMessageBox.Show();
                        if (x == DialogResult.OK)
                        {
                                ((GrammarRuleCollection)this.grammarTreeView.Model.Root).Save();
                                this.grammarTreeView.Reset();
                                new Grammar().ResetProductionRules();
                        }
                        else
                        {
                                this.Close();
                        }
                }

                // Bij het klikken op de save-button wordt de gehele gemuteerde
        productieregelcollectie
                // weggeschreven naar de database
                private void cbSave_Click(object sender, System.EventArgs e)
                {
                        ((GrammarRuleCollection)this.grammarTreeView.Model.Root).Save();
                        this.grammarTreeView.Reset();
                        new Grammar().ResetProductionRules();
                }

                private void cbPrint_Click(object sender, System.EventArgs e)
                {
                        DialogResult ok = this.PrintTo.ShowDialog();
                        if (ok == DialogResult.OK)
                        {
                                ((GrammarRuleCollection)this.grammarTreeView.Model.Root)
                        .Print(this.PrintTo.FileName);
                        }
                }
           }
        }
```

Workbench.NaturalLanguage.Grammar.Editor.GrammarTreeModel.cs

```
////////////////////////////////////////////////////////////////
/// WORKBENCH                                                 ///
///                                                           ///
///                                                           ///
///                                                           ///
/// © Copyright Belastingdienst (http://www.belastingdienst.nl)///
////////////////////////////////////////////////////////////////
/// Revision information:                                     ///
///   $Workfile:: GrammarTreeModel.cs                       $ ///
///   $Revision:: 1                                         $ ///
///   $Author:: Ron_van_gog, Kamal_Sayah                    $ ///
///   $Date:: 24/02/04                                      $ ///
////////////////////////////////////////////////////////////////
using System;
using System.Collections;

namespace Workbench.NaturalLanguage.Grammar.Editor
{
   /// <summary>
   /// Summary description for GrammarTreeModel.
   /// </summary>

   // Deze klasse zorgt voor de creatie van de TreeStructure uit het opgeslagen klassemodel
   public class GrammarTreeModel : Belastingdienst.Windows.Forms.BaseTreeModel
   {
          GrammarRuleCollection root;

          public void SetRoot(GrammarRuleCollection obj)
          {
                root = obj;
          }
          public override object Root
          {
                get
                {
                      return root;
                }
          }

          // Het vullen van alle kinderen van elk GrammarElement,GrammarFeature etc.
          public override System.Collections.IEnumerable ChildrenOf(object item)
          {
                ArrayList Result = new ArrayList();
                if (item is GrammarRuleCollection)
                {
                      GrammarRuleCollection ruleCollection = (GrammarRuleCollection) item;
                      foreach(GrammarRule temp in ruleCollection.RuleCollection)
                      {
                            if (!temp.isDeleted)
                            {
                                  Result.Add(temp);
                            }
                      }
                }
                else if (item is GrammarRule)
                {
                      GrammarRule rule = (GrammarRule) item;
                      Result.Add(rule.Lhs);
                      foreach(GrammarElement temp in rule.Rhs)
                      {
                            Result.Add(temp);
                      }
                }
                else if (item is GrammarElement)
                {
                      GrammarElement element = (GrammarElement) item;
                      foreach(GrammarFeature temp in element.FeatureSet)
                      {
                            Result.Add(temp);
                      }
                }
                else if (item is GrammarFeature)
                {
                      GrammarFeature feature = (GrammarFeature) item;
                      foreach(GrammarFeature temp in feature.FeatureSet)
                      {
                            Result.Add(temp);
                      }
                }
                return Result;
          }

          // Het genereren van de Stringrepresentaties van elke element
          public override string TextOf(object item)
          {
```

```
                if (item is GrammarRuleCollection)
                {
                        return "Rule Collection";
                }
                else if (item is GrammarRule)
                {
                        string Result = "";
                        GrammarRule rule = (GrammarRule) item;
                        Result = rule.Name + " [" + rule.Lhs.Name + " =>";
                        foreach (GrammarElement element in rule.Rhs)
                        {
                                if (element.Optional)
                                {
                                        Result += " (" + element.Name + ")";
                                }
                                else
                                {
                                        Result += " " + element.Name;
                                }
                        }
                        return Result + "]";
                }
                else if (item is GrammarElement)
                {
                        GrammarElement element = (GrammarElement) item;
                        string Result = element.Name;
                        if (element.Optional)
                        {
                                Result = "(" + Result + ")";
                        }
                        if (element.Lhs)
                        {
                                Result = "LHS " + Result;
                        }
                        else
                        {
                                Result = "RHS " + Result;
                        }
                        return Result;
                }
                else if (item is GrammarFeature)
                {
                        GrammarFeature feature = (GrammarFeature) item;
                        string Result = feature.Name;
                        if (feature.EquationId > -1)
                        {
                                Result += "[" + feature.EquationId.ToString() + "]";
                        }
                        if (feature.Atomic)
                        {
                                Result += " = " + feature.FeatureValue;
                        }
                        return Result;
                }
                return "???";
        }

        // Het aangeven of een element een leaf is of niet
        public override bool Leaf(object item)
        {
                if (item is GrammarRuleCollection)
                {
                        return false;
                }
                else if (item is GrammarRule)
                {
                        return false;
                }
                else if (item is GrammarElement)
                {
                        return false;
                }
                else if (item is GrammarFeature)
                {
                        GrammarFeature feature = (GrammarFeature) item;
                        if (feature.Atomic)
                        {
                                return true;
                        }
                        else
                        {
                                if (feature.FeatureSet.Count > 0)
                                {
                                        return false;
                                }
                                else
                                {
                                        return true;
                                }
                        }
```

```
176                             }
177                             return false;
178                         }
179                     }
180                 }
```

Appendix F

The following table contains a global view of the complete set of translation patterns (written in Visual Basic). These patterns are used for the translation of the parsed constructs into a formal model.

The patterns for the noun phrase extraction are based on Ron van Gog [37]. Note that some of the rules have been modified by the author.

Translation Patterns for Noun-phrase Extraction

Id	Condition	Script
	type = "np" and (root not in ("bedrag", "waarde", "hoogte") or pp.prep <> "van")	dim Result as New System.Collections.ArrayList dim Temp as Object Temp = Feature.Model.GetType(Feature.Item("root").ToString) Feature.Item("adj").Translate(Temp) Feature.Item("pp").Translate(Temp) Feature.Item("modif").Translate(Temp) Result.Add(Temp) return Result
	type = "adj_list"	Dim Result as New System.Collections.ArrayList Dim Temp as Object Dim Counter as Object Temp = Feature.Item("tl").Translate(Parent) for each Counter in Temp Result.Add(Counter) next Temp = Feature.Item("hd").Translate(Parent) for each Counter in Temp Result.Add(Counter) next return Result

	type = "adj"	Dim Result as New System.Collections.ArrayList
		dim Current as Object
		dim Temp as Object
		dim Counter as Object
		
		if Feature.Item("adv.hd.type").ToString.ToLower = "pp"
		Current = Feature.Item("adv.hd.main").Translate(Nothing)
		for each Counter in Current
		Temp = Feature.Model.GetAssociation(Parent, Counter,
		Feature.Item("adv.tl").Translate(Parent) + Feature.Item("root").ToString +
		Feature.Item("adv.hd.prep").ToString)
		Result.Add(Temp)
		next
		else
		Current = Parent.GetAttribute("Boolean",
		Feature.Item("adv").Translate(Parent) + Feature.Item("root").ToString)
		Result.Add(Current)
		end if
		
		return Result

	type = "pp"	dim Result as New System.Collections.ArrayList dim Temp as Object dim Current as Object dim Counter as Object Current = Feature.Item("main").Translate(Nothing) for each Counter in Current Temp = Feature.Model.GetAssociation(Parent, Counter, Feature.Item("prep").ToString) Result.Add(Temp) next return Result
	type = "adv_list"	dim Result as Object Result = Feature.Item("tl").Translate(Parent) + Feature.Item("hd").Translate(Parent) return Result
	type = "adv"	dim Result as Object Result = Feature.Item("main").ToString return Result
	type = "np" and root in ("bedrag", "waarde", "hoogte") and pp.prep = "van"	dim Result as New System.Collections.ArrayList dim Temp as Object dim Counter as Object dim Attr as Object Temp = Feature.Item("pp.main").Translate(Result) for each Counter in Temp Attr = Counter.GetAttribute("Real", Feature.Item("root").ToString) Result.Add(Attr) next return Result
	type = "bijvoeglijke_bijzin" and main.adv.hd.type = "pp"	Dim Result as New System.Collections.ArrayList dim Current as Object dim Temp as Object dim Counter as Object dim Name as String Current = Feature.Item("main.adv.hd.main").Translate(Nothing) Name = Feature.Item("main.pred.finit.main").ToString Name = Name + Feature.Item("main.adv.tl").Translate(Nothing) Name = Name + Feature.Item("main.pred.hoofd.main").ToString Name = Name + Feature.Item("main.adv.hd.prep").ToString for each Counter in Current Temp = Feature.Model.GetAssociation(Parent, Counter, Name) Result.Add(Temp) next return Result
	type = "np_money"	dim Result as New System.Collections.ArrayList dim Temp as Object Temp = Feature.Model.GetType(Feature.Item("cur").ToString + Feature.Item("root").ToString) Result.Add(Temp) return Result

	type="np_conj"	dim Result as New System.Collections.ArrayList dim Temp as Object dim Counter as Object Temp = Feature.Item("s1").Translate(Nothing) for each Counter in Temp Feature.Item("modif").Translate(Counter) Result.Add(Counter) next Temp = Feature.Item("s2").Translate(Nothing) for each Counter in Temp Feature.Item("modif").Translate(Counter) Result.Add(Counter) next return Result
	type = "np_ref"	dim Result as New System.Collections.ArrayList dim Temp as Object Temp = Feature.Model.GetPackageReference(Feature.Item("main").ToString) Result.Add(Temp) return Result
	type = "pp_conj"	dim Result as New System.Collections.ArrayList dim Temp as Object dim Counter as Object Temp = Feature.Item("s1").Translate(Parent) for each Counter in Temp Result.Add(Counter) next Temp = Feature.Item("s2").Translate(Parent) for each Counter in Temp Result.Add(Counter) next return Result

	type = "adj_conj"	Dim Result as New System.Collections.ArrayList
		Dim Temp as Object
		Dim Counter as Object
		Dim Adv as String
		
		Temp = Feature.Item("s1").Translate(Parent)
		for each Counter in Temp
		Result.Add(Counter)
		next
		
		Temp = Feature.Item("s2").Translate(Parent)
		for each Counter in Temp
		Result.Add(Counter)
		next
		
		Adv = Feature.Item("adv").Translate(Nothing)
		If Adv <> "" then
		Adv = Adv.Substring(0,1).ToLower + Adv.Substring(1)
		for each Counter in Result
		Counter.Name = Adv + Counter.Name.Substring(0,1).ToUpper +
		Counter.Name.Substring(1)
		next
		end if
		
		return Result

	type = "bijvoeglijke_bijzin" and main.adj.type = "adj"	Dim Result as New System.Collections.ArrayList Dim Temp as Object Temp = Feature.Item("main.adj").Translate(Parent) Result.Add(Temp) return Result
	type = "pp2"	dim Result as New System.Collections.ArrayList dim Counter as Object Result = Feature.Item("main").Translate(Parent) for each Counter in Result Counter.Name = Feature.Item("prep").ToString.ToLower + Feature.Item("main.prep").ToString next return Result

Translation Patterns for Verb-phrase Extraction

Id	Condition	Script

	type = "s_dp"	dim Result as New System.Collections.ArrayList
		dim EC as New System.Collections.ArrayList
		dim Counter as Object
		dim AttrCounter as Object
		dim EcCounter as Object
		dim Attr as Object
		dim strCondition as String
		dim strTemp as String
		
		Result = Feature.Item("subject").Translate(Nothing)
		for each Counter in Result
		strCondition = ""
		for each AttrCounter in Counter.myAttributes
		if strCondition = "" then
		strCondition = AttrCounter.Name
		else
		strCondition = strCondition + " and " + AttrCounter.Name
		end if
		next
		
		if Feature.item("ec.type").ToString <> "" then
		EC = Feature.Item("ec").Translate(Nothing)
		for each EcCounter in EC
		for each AttrCounter in EcCounter.myAttributes
		if Counter.Name = EcCounter.Name then
		strTemp = AttrCounter.Name
		else
		strTemp = EcCounter.Name + "." + AttrCounter.Name
		end if
		if strCondition = "" then
		strCondition = strTemp
		else
		strCondition = strCondition + " and " + strTemp
		end if
		next
		next
		end if
		
		if strCondition <> "" then
		strCondition = strCondition + " implies "
		end if
		
		Attr = Counter.GetAttribute("Boolean", Feature.Item("dp_part1").ToString +
		Feature.Item("time_period").Translate(Nothing) +
		Feature.Item("dp_part2").ToString +
		Feature.Item("fiction").Translate(Nothing))
		
		Counter.GetConstraint("attributeInvariant", strCondition + Attr.Name)
		next
		
		Return Result

	type = "x_list"	dim Result as Object Result = Feature.Item("hd").ToString + Feature.Item("tl").Translate(Parent) return Result
	type = "bijvoeglijke_bijzin" and main.type = "x_list"	Dim Result as New System.Collections.ArrayList Dim myAttribute as Object Dim AttrName as Object AttrName = Feature.Item("main").Translate(Parent) myAttribute = Parent.GetAttribute("Boolean", AttrName) Result.Add(myAttribute) return Result

	type = "ec"	dim Result as New System.Collections.ArrayList dim Counter as Object Result = Feature.Item("subject").Translate(Nothing) for each Counter in Result Counter.GetAttribute("Boolean", Feature.Item("feature").Translate(Nothing)) next Return Result

	type = "s_def"	dim Result as New System.Collections.ArrayList
		dim Temp as New System.Collections.ArrayList
		dim Counter as Object
		dim Counter2 as Object
		dim Assoc as Object
		dim strConstraint as String
		dim boolFound as Boolean
		
		strConstraint = ""
		
		Result = Feature.Item("subject").Translate(Nothing)
		for each Counter in Result
		Counter.Name = Counter.Name + "*"
		next
		Temp = Feature.Item("direct_object").Translate(Nothing)
		for each Counter in Result
		boolFound = false
		for each Counter2 in Temp
		if Counter.Name = Counter2.Name + "*" then
		boolFound = true
		end if
		next
		if boolFound = false then
		Counter.Name = Left(Counter.Name, Len(Counter.Name) - 1)
		end if
		next
		
		for each Counter in Temp
		for each Assoc in Counter.myAttributes
		strConstraint = strConstraint + " and " + Assoc.Name
		next
		for each Assoc in Counter.myAssociations
		strConstraint = strConstraint + " and " + Assoc.Name + "->notEmpty"
		next
		if strConstraint <> "" then
		strConstraint = strConstraint.SubString(5)
		end if
		next
		
		for each Counter in Result
		Counter.Supertype = Temp(0)
		if strConstraint <> "" then
		Counter.GetConstraint("Invariant", strConstraint)
		end if
		next
		
		Result.Add(Temp)
		
		return Result

	type = "s_def2" or type = "s_def3" or type = "s_def4"	dim Result as New System.Collections.ArrayList
		dim Temp as New System.Collections.ArrayList
		dim SDef as New System.Collections.ArrayList
		dim EC as New System.Collections.ArrayList
		dim Counter as Object
		dim Counter2 as Object
		dim boolFound as Boolean
		dim EcCounter as Object
		dim AttrCounter as Object
		dim SDefCounter as Object
		dim Assoc as Object
		dim strConstraint as String
		dim strExtraConstraint as String
		dim strExtraConstraint2 as String
		dim strStereotype as String
		dim strTypeName as String
		dim strCondition as String
		dim StrTemp as String
		
		if Feature.item("ec.type").ToString <> "" then
		EC = Feature.Item("ec").Translate(Nothing)
		for each EcCounter in EC
		for each AttrCounter in EcCounter.myAttributes
		strTemp = EcCounter.Name + "." + AttrCounter.Name
		if strCondition = "" then
		strCondition = strTemp
		else
		strCondition = strCondition + " and " + strTemp
		end if
		next
		next
		end if
		
		Result = Feature.Item("subject").Translate(Nothing)
		for each Counter in Result
		Counter.Name = Counter.Name + "*"
		next
		Temp = Feature.Item("definition").Translate(Nothing)
		for each Counter in Result
		boolFound = false
		for each Counter2 in Temp
		if Counter.Name = Counter2.Name + "*" then
		boolFound = true
		end if
		next
		if boolFound = false then
		Counter.Name = Left(Counter.Name, Len(Counter.Name) - 1)
		end if
		next
		
		if Feature.Item("sdef.type").ToString <> "" then
		SDef = Feature.Item("sdef").Translate(Nothing)
		end if
		
		Counter = Result(0)
		strExtraConstraint = ""
		for each Assoc in Counter.myAttributes
		strExtraConstraint = strExtraConstraint + " and " + Assoc.Name
		next
		for each Assoc in Counter.myAssociations
		strExtraConstraint = strExtraConstraint + " and " + Assoc.Name + "->notEmpty"
		next
		
		if Feature.Item("adv").ToString.ToLower = "niet"
		strTypeName = "not self:" + Result(0).Name
		else
		strTypeName = "self:" + Result(0).Name
		end if
		
		if Feature.Item("adv").ToString.ToLower = ""
		strStereotype = "typeInvariant"
		else
		strStereotype = "typeExtensionInvariant"

```
end if

for each Counter in Temp
strConstraint = ""
for each Assoc in Counter.myAttributes
strConstraint = strConstraint + " and " + Assoc.Name
next
for each Assoc in Counter.myAssociations
strConstraint = strConstraint + " and " + Assoc.Name + "->notEmpty"
next
if strConstraint <> "" then
strConstraint = strConstraint.SubString(5)
end if

if Feature.Item("sdef.type").ToString <> "" then
for each SDefCounter in SDef
Feature.Model.GetAssociation(Counter, SDefCounter, "scopeDefinition")
if strConstraint <> "" then
strConstraint = "applies(" + SDefCounter.Name+ ") and " + strConstraint
else
strConstraint = "applies(" + SDefCounter.Name+ ")"
end if
next
end if

if Counter.Name = Result(0).Name then
strExtraConstraint2 = ""
else
strExtraConstraint2 = strExtraConstraint
end if

if strConstraint <> "" then
if strCondition<> "" then
strConstraint = strCondition + " and " + strConstraint + " implies " +
strTypeName + strExtraConstraint2
else
strConstraint = strConstraint + " implies " + strTypeName +
strExtraConstraint2
end if
else
if strCondition<> "" then
strConstraint = strCondition + " implies " + strTypeName +
strExtraConstraint2
else
strConstraint = strCondition + " implies " + strTypeName +
strExtraConstraint2
end if
end if

Counter.GetConstraint(strStereotype, strConstraint)
next

return Result
```

	type = "s_app"	dim Ref as New System.Collections.ArrayList
		dim EC as New System.Collections.ArrayList
		dim EcCounter as Object
		dim AttrCounter as Object
		dim strCondition as String
		dim strTemp as String
		dim strApplies as String
		dim Temp as Object
		
		Ref = Feature.Item("ref").Translate(Nothing)
		strCondition = ""
		
		if Feature.item("ec.type").ToString <> "" then
		EC = Feature.Item("ec").Translate(Nothing)
		for each EcCounter in EC
		for each AttrCounter in EcCounter.myAttributes
		strTemp = EcCounter.Name + "." + AttrCounter.Name
		if strCondition = "" then
		strCondition = strTemp
		else
		strCondition = strCondition + " and " + strTemp
		end if
		next
		Temp = Feature.Model.GetAssociation(EcCounter, Ref(0),
		"VanToepassingVerklaring")
		next
		end if
		
		if Feature.Item("adv").ToString.ToLower = "niet"
		strApplies = "not applies(" + Ref(0).Name + ")"
		else
		strApplies = "applies(" + Ref(0).Name + ")"
		end if
		
		if strCondition <> "" then
		strCondition = strCondition + " implies " + strApplies
		else
		strCondition = strApplies
		end if
		
		Ref(0).GetConstraint("invariant", strCondition)
		
		return Ref

	type="s_va"	dim Result as New System.Collections.ArrayList
		dim Temp as New System.Collections.ArrayList
		dim Invariant as String
		dim Attr as Object
		dim Counter as Object
		dim CounterTemp as Object
		
		Result = Feature.Item("subject").Translate(Nothing)
		
		if Feature.Item("formula.type").ToString.ToLower <> "np_formula" then
		Temp = Feature.Item("formula").Translate(Nothing)
		if Temp(0).GetType().Name = "PType" then
		Attr = Temp(0).GetAttribute("Real", "bedrag")
		else
		Attr = Temp(0)
		end if
		Invariant = Attr.ParentType.Name + "." + Attr.Name
		else
		Invariant = Feature.Item("formula").Translate(Nothing)
		end if
		
		if Result(0).GetType().Name = "PType" then
		Result(0).GetAttribute("Real", "bedrag")
		Result(0).GetConstraint("invariant", "bedrag" + " = " + Invariant)
		else
		Result(0).ParentType.GetConstraint("invariant", Result(0).Name + " = " +
		Invariant)
		end if
		
		return Result

	type="np_formula"	dim X as New System.Collections.ArrayList
		dim Y as New System.Collections.ArrayList
		dim Result as String
		dim AttrX as Object
		dim AttrY as Object
		
		Result = ""
		
		X = Feature.Item("x").Translate(Nothing)
		Y = Feature.Item("y").Translate(Nothing)
		
		if X(0).GetType().Name = "PType" then
		AttrX = X(0).GetAttribute("Real", "bedrag")
		else
		AttrX = X(0)
		end if
		
		if Y(0).GetType().Name = "PType" then
		AttrY = Y(0).GetAttribute("Real", "bedrag")
		else
		AttrY = Y(0)
		end if
		
		if Feature.Item("plusminus").ToString.ToLower = "vermeerderen" then
		Result = AttrX.ParentType.Name + "." + AttrX.Name + " + " +
		AttrY.ParentType.Name + "." + AttrY.Name
		else
		Result = AttrX.ParentType.Name + "." + AttrX.Name + " - " +
		AttrY.ParentType.Name + "." + AttrY.Name
		end if
		
		return Result

	type = "scopedef"	dim Result as New System.Collections.ArrayList Result = Feature.Item("ref").Translate(Nothing) return Result
	type = "s_rel"	dim Result as New System.Collections.ArrayList dim Temp as New System.Collections.ArrayList dim AssocCounter as Object dim Counter as Object Result = Feature.Item("subject").Translate(Nothing) for each Counter in Result Temp = Feature.Item("pp").Translate(Counter) for each AssocCounter in Temp AssocCounter.Name = Feature.Item("verb").ToString.ToLower + Feature.Item("pp.prep").ToString next next return Result

	type = "nabepaling"	Dim Result as New System.Collections.ArrayList
		Dim NP as New System.Collections.ArrayList
		Dim myAttribute as Object
		Dim myAssoc as Object
		Dim AttrName as Object
		Dim Counter as Object
		
		AttrName = Feature.Item("main").Translate(Parent)
		
		if Feature.Item("np.type").ToString = "" then
		myAttribute = Parent.GetAttribute("Boolean", Feature.Item("adv").ToString + AttrName)
		Result.Add(myAttribute)
		else
		NP = Feature.Item("np").Translate(Parent)
		for each Counter in NP
		if Counter.GetType().Name = "PType" then
		myAssoc = Feature.Model.GetAssociation(Parent, Counter, AttrName)
		Result.Add(myAssoc)
		end if
		next
		end if
		
		return Result