Morten Ib Nielsen

Logical Models for Value-Added Tax Legislation

– Progress Report –

October 1, 2008

Department of Computer Science
University of Copenhagen
Denmark

Preface

The document at hand is my progress report concluding part A (two years) of my four year Ph.D. program. I enrolled in the four year Ph.D. program (the most common duration of a Ph.D. program in Denmark being three years) after one year of master's studies, which means that I was only halfway through my graduate program at the time and furthermore I had (and still have) not obtained my master degree.

The purpose of the progress report is to present a snapshot of the (ongoing and unfinished) research I am currently involved in and to account for the course of actions that have led us there. Consequently the form of a progress report can be somewhat different from that of a master thesis (Danish: speciale). For instance the progress report does not need to present final conclusions (it reports on unfinished research), nor does it need to be self contained (i.e., it is allowed to have open ends).

This progress report consists of five documents. The first, and most important one, documents our most recent research as well as activities that have not been documented before. The remaining four documents have been added as appendices to the first document. Readers should begin with the first document, and consult appendices as they see fit in order to shed light on the context.

Below I present the appendices in order of creation together with their current state and I list coauthors:

Tutorial on Modeling VAT Rules Using OWL-DL. Submitted to and presented at the first 3gERP workshop, October 21-22 at Copenhagen Business School. Coauthored with Jakob Grue Simonsen and Ken Friis Larsen (Department of Computer Science, University of Copenhagen).

A System of VAT Rules. Remains unpublished.

Modeling VAT Rules in CPML. Remains unpublished. Includes details that are not present in the next item.

Logical Models for Value-Added Tax Legislation. Submitted to IWIL08. Coauthored with Jakob Grue Simonsen and Ken Friis Larsen (Department of Computer Science, University of Copenhagen).

Copenhagen
December 10, 2008

Morten Ib Nielsen

Resumé

Denne rapport behandler logisk modellering af juridiske regler med henblik på anvendelse i ERP-systemer. Rapporten byggere videre på de i forordet nævnte arbejder. For yderligere beskrivelse henvises til det engelske resumé (abstract).

Abstract

This report discusses logical modeling of legal rules in the context of Enterprise Resource Planning (ERP) systems. The main contributions are: an analysis of the legal domain from an ERP perspective (including a description of reasoning in the legal domain), an account of how state of the art ERP systems (NAV and SAP) support VAT, and a suggestion of a domain specific language for legal rule modeling. We conclude that VAT support rendered by current ERP systems suffer from several defects such as oversimplification and lack of causality between VAT code and supported legal rules to name a few.

The report has two parts. Part one contains the analysis of the legal domain and the account of the VAT support rendered by state of the art ERP systems. This leads to a list of requirements for the design of a domain specific language intended to enable domain experts to model legal rules. In the second part we discuss an early suggestion for such a language through examples and we outline directions for future work.

Acknowledgements

First and foremost I owe much to my supervisors Fritz Henglein and Ken Friis Larsen without whom this work would not have been possible. They have been invaluable sources of information on all sorts of things, but most importantly they continue to provide a good working environment professionally and personally.

I would also like to thank Kim Graversen[1] (SAP consultant at Applicon A/S) for explaining how SAP handles VAT, and for pointing me to relevant reading material on the subject. Similarly I would like to thank cand.jur Sabrina Sahl[2] (officer at the Danish National Tax Tribunal) and stud.jur Martin Henrik Jensen[3] for fruitful discussions about legal issues especially reasoning and exceptions to rules. Without their efforts I would not have been able to give an accurate account.

Furthermore I would like to thank Jakob Dirksen[4] and Mikkel Stein Knudsen[5] for non-technical feedback on my presentation and for suggesting improvements. I also want to thank my colleagues for the discussions we have had and for our good friendship, which is very important to me.

Finally I want to state that whatever errors and imperfections that might be left are mine and mine alone.

Thank You!

[1] kig@applicon.dk.
[2] ss@lsr.dk.
[3] maje507@student.sdu.dk.
[4] j_dirksen@hotmail.com.
[5] mikkelsknudsen@gmail.com.

Contents

Preface III

Resumé V

Abstract VII

Acknowledgements IX

Part I Legal Rules in an ERP Context

1 Introduction 3
- 1.1 A View From the Inside 4
- 1.2 Towards a Solution 5
  - 1.2.1 First Steps 6
- 1.3 Moving Forward 7
  - 1.3.1 Assumptions 9
  - 1.3.2 Work Ahead 9
- 1.4 Outline 9

2 The Legal Domain 11
- 2.1 Structure 11
- 2.2 Concepts and Rules 12
- 2.3 Legal Doctrine 14
  - 2.3.1 Interpreting Exceptions 15
- 2.4 Summary 17

3 Support for Legal Rules in ERP Systems 19
- 3.1 Microsoft Dynamics NAV 19
  - 3.1.1 Quotes and orders 20
  - 3.1.2 Example 23

3.1.3 Summing up 23
3.2 SAP 24
3.2.1 Finding the Tax Code 24
3.2.2 How to set up VAT Schemes 26
3.2.3 Summing up 28
3.3 Comparison and Summary 29
3.3.1 Lack of Transparency is the Problem 29

4 Requirements on a Model 31
4.1 The Isomorphism Principle 31
4.2 Tractable Analysis vs. Expressiveness 32
4.2.1 Concepts 32
4.2.2 Structure of Rules, Operations and Workflows 33
4.3 Legal Reasoning and Collision Rules 34
4.4 Tool Support and Usage 35

Part II LLMBRR

5 Discussion 41
5.1 LLMBRR Syntax 42
5.1.1 Explaining the Syntax 42
5.1.2 The Meaning of **this** 44
5.1.3 Simplifying Assumptions 45
5.2 Examples 45
5.2.1 Encoding VAT Rules From Microsoft Dynamics NAV 46
5.2.2 Encoding Guidelines 47
5.3 Legal Reasoning in the Context of LLMBRR 50
5.3.1 Comments 52

6 Summary & Future Work 53
6.1 Summary 53
6.2 Looking Forward 54

A Well-formed LLMBRR 55

B Example Code 61
B.1 LLMBRR Encoding of VAT Rules From Microsoft Dynamics NAV 61
B.1.1 Straight Forward Encoding 61
B.1.2 Encoding Using Dependencies Between Values 62
B.2 LLMBRR Encoding of VAT on Danish Services 63

C E-mail correspondance 67
C.1 First Mail from Kim Graversen 67
C.2 Second Mail from Kim Graversen 68
C.3 Third Mail from Kim Graversen 69
C.4 Fourth Mail from Kim Graversen 70

References 73

Part I

Legal Rules in an ERP Context

1 Introduction

Whenever you buy something, whether it is in the supermarket, via the web or in any other kind of shop, you have to pay value added tax (VAT). Simple as that - right? In fact it is not given at all that you have to pay VAT, and if you do maybe a reduced rate applies to the kind of item you are buying. As consumers we usually leave issues such as calculation of VAT to the shopkeepers (or business owners), but what do they actually do when addressing the question of VAT? Today, in most cases, the answer is that they rely on a computerized system running a financial application customized to their specific needs. And they have good reason to do so because VAT is not simple at all! As an illustration let us consider the following: first of all, in Europe, VAT is harmonized in *COUNCIL DIRECTIVE 2006/112/EC* [5], a 118 pages long legal document describing the frame within which the individual member states must implement their VAT laws. Secondy we can look at the size of official guides written for professionals such as accountants and lawyers. In Denmark, where the VAT system is rather simple compared to other European countries, this guide [10] is more than a 1000 pages long.

It turns out that shopkeepers are not the only ones relying on computerized systems to calculate VAT correctly. To some extent auditors, whose job it is to make sure that financial transactions are made in accordance with the law, do the same in practise and consequently they might be less scrutinizing, when they examine books produced by computerized systems than they would have been otherwise.

For these reasons it is important to know what computerized financial systems do. How do they calculate VAT and how can we be (or make) sure they do so correctly? The answer to these questions depends on what system you use. In this report we examine Microsoft Dynamics NAV and SAP two major Enterprise Resource Planning (ERP) systems. Described shortly ERP systems are humongous software systems that in principle facilitate control with every aspect of a business (e.g., sales, inventory, production, HR, BI, accounting etc.).

In order to answer the questions we must change our point of view.

1.1 A View From the Inside

From interviews with Microsoft staff we know that handling of VAT is an important and complex issue in Microsoft Dynamics NAV and AX. Here we shall focus mostly on NAV, but many of the arguments carry over to other ERP systems as well. In NAV complexity stems from the large number of countries with (slightly) different VAT schemes that are supported. This taken together with the fact that handling of VAT today is a cross-cutting feature (i.e., the VAT code is spread throughout the NAV code base) makes it hard, time consuming, and thus expensive to maintain VAT functionality.

In addition the nature of the process used to adapt the system when changes in the underlying VAT legislation occur (or when new features are needed) also contributes significantly to the maintenance time (and cost). When changes are needed a chain of specialists, ranging from domain experts (e.g., accountants and lawyers) on VAT to programmers, must create an informal program/feature specification. This is done in a stepwise sequential manner where each step refines and takes the specification closer to something programmers can (understand and) implement.

Another concern, stemming from the cross-cutting nature of VAT code, which is amplified by the fact that ERP systems evolve over time (and that documentation is not always as good as one could wish for) is that it is hard if not impossible to determine exactly what parts of the VAT legislation the system supports. We shall argue that ERP systems often implement only the most general rules, which we can think of as conservative approximations (the authorities do not mind that you charge and pay VAT you did not have to), and in the case of NAV Microsoft leaves it to partners to customize the system if nonstandard functionality is needed.

In summary computerized handling of VAT poses the following problems:

1. How can we be or make sure VAT is calculated correctly?
2. It is hard if not impossible to know exactly what rules the system supports.
3. It is complex due to the large number of countries and rules.
4. The cross-cutting nature makes it hard, time consuming and expensive to maintain.
5. Changes require involvement of a long chain of people.
6. Partners need to provide customized solutions, when nonstandard functionality is needed.

Some of the problems are only relevant to the vendor of the ERP (or financial) system (i.e., items 3-6), while others are relevant to several parties. For instance the first item is relevant to the consumer, because she wants to acquire her products as cheap as possible; it is relevant to the shopkeeper, because he wants to profit as much as possible, while maintaining competitive advantage over his competitors; and finally it is relevant to the auditor because she must make sure the books produced by the system do not violate the law.

1.2 Towards a Solution

In our opinion the two most important steps to take in order to address the problems outlined above is to support separation of concerns by gathering all VAT functionality in one place (e.g., a module) and to provide easy navigation between rules in legal documents and the code implementing the rules and back. This would help to solve the problem with the cross-cutting nature of VAT code as well as making it easier to identify what parts of the VAT law the system supports, while at the same time make it easier for partners to create extensions and add-ons. In addition better support for complexity could be achieved in part by specifying rules in a declarative fashion closer to the way they are formulated in legal documents.

But what about the first and the fifth problem? If we begin with the first problem, then how can we ensure that VAT is calculated correctly for a particular item? Well, in some sense it is impossible because the information about the item in the system might be wrong. The distinction between what is true in the *real world* and what is known in the system is interesting in its own right, and we will get back to that in Section 1.3. But if we ignore this distinction what we seek is evidence (a proof) of why the system did as it did. Regarding the fifth problem we would like to find a way to shorten the chain of people involved, which essentially means that we need people closer to the legal end of the chain to implement the rules.

We believe all of the above can be realised through a model driven approach to legal rules for ERP systems based on or inspired by decidable logic. By model driven approach we mean the formal specification of functional requirements (legal rules) as platform-independent models (PIM) using a domain specific language (DSL).

The first problem could be solved by requiring the (runtime interpretation of the) model to produce a proof (a derivation) of how it reaches its conclusions. If the modeling language is specific to the legal domain we can assume that it provides better support for complexity and makes it easier to get an overview of the implemented rules as well as maintaining and extending the model. In turn this would contribute to lowering the time and thus the price of adapting the model. Furthermore domain experts could be educated to use the language or paired with rule engineers in small teams ultimately shortening the chain of people involved by implementing legal rules directly, and thus obviating the need to create informal specifications whenever changes are due. Another benefit from this approach, assuming that domain experts are directly involved in producing the model, is that we remove several layers of human interpretation, and thus a possibility to introduce bugs due to misinterpretation.

In addition to the benefits outline above we would also gain the option of applying automated analysis to models. For instance we could check for inconsistency, do dead rule elimination, simulate effects of small changes more easily etc. Finally, one can imagine that several VAT models exist (e.g., one for each country or region). If we consider this together with the fact that a VAT model should describe the entire behaviour of the system with respect to VAT we see that a model driven approach changes VAT support from being a customization task into a configuration issue.

In summary we find that a model driven approach to VAT has the following benefits:

1. Promotes separation of concerns.
2. Auditors can test the calculations produced by the system by validating accompanying proofs against rules in the VAT legislation.
3. Supports complexity better and provides better overview of what rules the system implements.
4. Obviates the need for informal specifications, whenever changes are due.
5. Removes several layers of human interpretation leading to fewer bugs.
6. Ultimately shortens development and maintenance time and cost.
7. Models are amenable to automated analysis.
8. VAT support becomes a configuration issue instead of a customization task.

Based on the above we state the following thesis:

Thesis: A model driven approach to legal rules for ERP systems based on decidable logic is feasible and useful, where useful means that the approach is at least as good as the state of the art and in addition contributes with new possibilities such as static analysis.

In the next section we describe our initial investigations briefly in order to shed light on the context and to set the scene for the research reported in the remainder of this document.

1.2.1 First Steps

We decided to look at OWL-DL as a first approximation of a modeling language because it is expressive, easy to use (via Protégé [4]) and reasoners exist (e.g., Racer Pro [19]).

OWL

OWL, which is short for Web Ontology Language, is an ontology language developed to support the semantic web. It comes in three variants: OWL-Lite $\subset$ OWL-DL $\subset$ OWL-Full of increasing expressive power. The variants OWL-Lite and OWL-DL are based on the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ respectively [6], which guarantees decidability of important inference problems such as satisfiability and subsumption.

The most important abstraction in OWL is classes, which we can think of as sets. Each class has a list of necessary conditions and zero or more equivalent lists of necessary and sufficient conditions for membership of the class [9]. Using different operators it is possible to require (members of) classes to have attributes and in addition new classes can be constructed from existing ones (e.g., using union and intersection operators) and classes can be asserted to be non overlapping.

Using OWL we developed a model of a subset of the Danish VAT rules describing exempt supplies and we have established a modeling methodology as well [12]. In conclusion we found that OWL lacked support for integer inequalities and that reasoning was not supported for certain data type properties.

A Systematic Approach

Having completed the experiments with OWL we felt the need to take a step back and look at our goal in a more systematic way before we went on to try another formalism. This led to the creation of a small but representative collection of VAT rules relevant in an ERP context [11]. Here relevant means that the rules (ought to) influence the way ERP systems handle VAT and representative means that a formalism able to model the rules ought to be able to model any relevant VAT rule.

The rules were intended to lay the foundations for benchmarking and performing a systematic comparison of (the expressiveness of) different approaches to VAT modeling.

CPML

Next off we decided to try the Configit Software Suite [2], which is a commercial product designed to build interactive sales and product configurators. The suite contains a graphical tool for building CPML models, which consist of a set of variables over finite domains together with (logical) rules, relations and functions on them. CPML can be regarded as a constraint logic programming language since rules and relations specify interrelationships between variables in first order propositional logic. We decided to try CPML over languages such as Prolog and RuleML because of the great tool support and because it supports full inference.

Using Product Modeller we created a model of most of the rules in *A System of VAT Rules* [11]. In conclusion we found that CPML is expressive enough to handle non-temporal VAT rules, but also that Product Modeller is not suitable for modeling of a full scale set of VAT rules due to lack of domain support for the legal domain.

1.3 Moving Forward

The goal at hand is to alleviate the problems outlined in Section 1.1 in order to obtain the benefits summarized in Section 1.2. As mentioned we believe the right way to do so is to provide domain experts with a DSL allowing them to implement VAT rules for ERP themselves.

In Figure 1.1 we present an overview of how we envision a model of VAT rules could be used to drive a VAT query engine, which in turn could be used to take decisions about VAT in ERP systems. We note that the ERP system and the query engine are loosely coupled through a query API, which promotes separation of concerns and in addition makes it easy to use a model for other purposes.

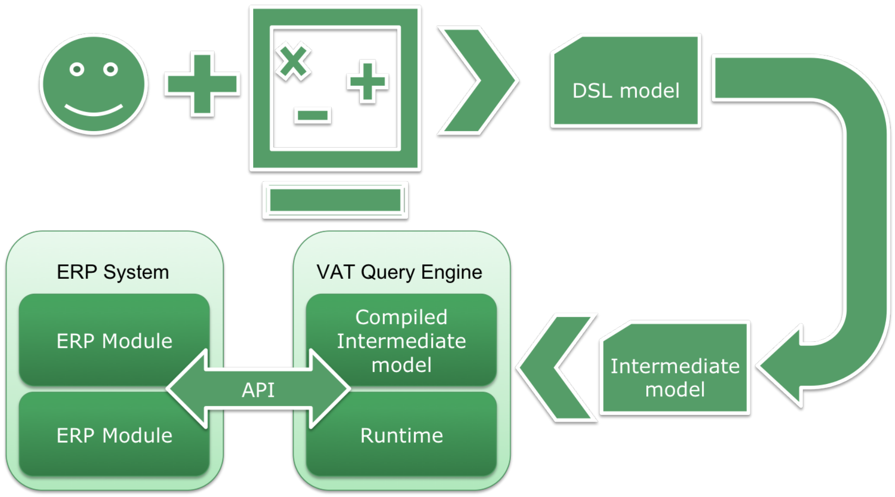


Fig. 1.1. Integration of ERP System and VAT Query Engine.

Formalizing legal rules is not a new idea. An early example is the formalisation of the British Nationality Act by Sergot et al. [17]. The goal of such systems was (in principle) to replace domain experts, which Leith [7] has later argued is not feasible due to what Prakken [15] refers to as the *open texturedness of legal concepts*.

Open texture as presented by Prakken [15] is the problem of interpreting legal concepts including the mapping of real world objects to legal concepts. Prakken [15, p. 49-55] identifies four kinds of open texturedness: *underdetermination*; the situation where it is not clear whether an instance is a member of a concept, *overdetermination*; where conflicting rules exist, *defeasibility of legal concepts*; the situation where unspecified (unforeseen) exceptions might exist and *vagueness*; covering rules using terms such as reasonable, sufficient, and so on.

Prakken [15, p. 59] suggests the use of non-monotonic logics to handle problems of open texture as well as the problem of separation of main rules and exceptions (we get back to this later). An immediate consequence of using non-monotonic logic is that all of the knowledge base (KB) must be searched for every query.

It is important to understand that our goal is not to replace domain experts, but merely to alleviate the problems outlined in Section 1.1. As mentioned in Section 1.2 we believe our approach has several benefits, one of which is to lower development and maintenance cost in connection with support for VAT. This will allow ERP vendors to increase their profit or to lower the cost of their software. In the last case it would lead to a reduced Total Cost of Ownership (TCO) for ERP customers (shopkeepers), which is one of the goals of the strategic research project 3gERP (under which the work on this project has been carried out).

1.3.1 Assumptions

In order to resolve problems of open texture we shall assume that domain experts resolve underdetermination, that defeasibility of legal concepts does not occur and that vagueness is resolved either statically during modeling or through dialogue with users. However we shall look at how to handle the situation, where conflicting rules exist.

1.3.2 Work Ahead

In searching the literature we have not been able to find a DSL supporting formal modeling of legal rules. Nor have we been able to find any analysis describing the requirements for such a language. Therefore we have decided to do the analysis and to develop a DSL ourselves.

1.4 Outline

This report is divided into two parts. Following this introduction we continue the first part with an analysis of legal rules in an ERP context with special focus on VAT. The goal is to establish background knowledge with respect to the legal domain: to find out what legal rules look like, and how practitioners reason; and to describe how state of the art systems such as Microsoft Dynamics NAV and SAP currently handle VAT (and legal rules in general); and to identify and characterize potential problems. This leads to a list of requirements on the design of a DSL for legal modeling. In part two we discuss how a DSL can be designed based on the requirements identified in part one. The thoughts and views presented in part two are not final and thus they are subject to change as our research moves forward. Part two ends with a summary of the work done so far and an outline of directions for future work.

2

The Legal Domain

In this chapter we describe features of the legal domain that our initial investigations have suggested are important with respect to modelling given the goal we presented in Section 1.3. Our description is mainly based on the European Directive on the common system of value added tax [5], but it also applies to country specific VAT acts such as the British [21] and the Danish [18].

2.1 Structure

From a top-level perspective legal documents are structured collections of uniquely identifiable pieces of natural language text written in legal vernacular. Here we describe the structure as it occurs in the directive [5]. Our description shall take the notion of an *article* as its starting point.

Articles in the directive are uniquely identifiable (sequentially numbered from 1) and are grouped using the following constructions: *TITLE*, *CHAPTER*, *Section* and *Subsection*. The grouping rules can be described using the following BNF [1], where *dir* is a directive:

$$\begin{aligned} dir &::= TITLE^{1+} \\ TITLE &::= CHAPTER^{2+} \mid article^{1+} \\ CHAPTER &::= Section^{2+} \mid article^{1+} \\ Section &::= Subsection^{2+} \mid article^{1+} \\ Subsection &::= article^{1+} \end{aligned}$$

Seen this way a directive is a tree structure where the articles are leafs and the grouping constructions are nodes on different levels. The grouping constructions are used for two related purposes. One is to be able to reference

[1] We use $X ::= T^{n+}$ as a shorthand notation for $X ::= T'$, $T' ::= \underbrace{T \cdots T}_{n} | TT'$.

a collection of articles namely the ones having a given grouping construction as an ancestor, the other is to group articles in a (to humans) meaningful way. To this end grouping constructions carry headlines (short descriptions) as well as identification numbers, but while articles are uniquely numbered on a directive wide basis grouping constructions are numbered sequentially from 1 within their enclosing construction (e.g., there can be a chapter 2 of title 1 as well as a chapter 2 of title 2 etc).

Hierarchical structure can also be imposed within individual articles. If we take *lstmt* (short for legal statement) to be a piece of legal vernacular without any further (hierarchical) structure we can describe the intra article grouping constructions and rules using the following BNF:

$$\begin{aligned} article &::= paragraph^{2+} \;\big|\; lstmt^{1+} \\ paragraph &::= schedule^{2+} \;\big|\; lstmt^{1+} \\ schedule &::= point^{2+} \;\big|\; lstmt^{1+} \\ point &::= lstmt^{1+} \end{aligned}$$

The intra article grouping constructions are enumerated in a fashion similar to chapters, sections etc. and serve the same purposes (referencing and grouping of related legal statements).

Being able to reference (collections of) articles is important since the provisions of (part of) one article is often subject to the content of another. Similar to many other situations (e.g., file systems) references can be absolute or relative. Absolute references begin with either a title or an article number while relative references are (big surprise) relative to the place in which they occur. As an example consider Article 2 in the directive [5]. It consists of three paragraphs each of which has several schedules and points. In paragraph 2 schedule (a) a reference is given as follows: *For the purposes of point (ii) of paragraph 1(b),...*. This is an example of a relative reference to the same article, but a different paragraph, schedule and point.

Another issue related to structure and references is the common separation of (main) rules and their exceptions. The separation comes about because of the way in which legal documents evolve over time due to the fact that legislators cannot foresee all possible futures/usages, see Prakken [15] for an elaborate discussion.

2.2 Concepts and Rules

Now that we have described the structure of legal documents we are ready to describe their content. Not all content is relevant in an ERP context. An example is rules specifying procedures for updating the legislation such as Article 8 in the directive [5]. As outlined in the introduction our goal is to model rules that are *relevant* in an ERP setting. In order to do this we need to identify the (different kinds of) relevant rules and to develop a *modelling methodology.* In order to leverage both of these tasks we have undertaken an enquiry into the nature of legal statements based on the directive [5], which has resulted in a classification scheme for rules[2] of which we present the relevant parts below:

Definitions are legal statements introducing new concepts which can be used in rules. Definitions can be either explicit such as the definition of *Taxable person*:

> 'Taxable person' shall mean any person who, independently, carries out in any place any economic activity, whatever the purpose or results of that activity.
> Any activity of producers, traders or persons supplying services, including mining and agricultural activities and activities of the professions, shall be regarded as 'economic activity'. The exploitation of tangible or intangible property for the purposes of obtaining income therefrom on a continuing basis shall in particular be regarded as an economic activity.
> [5, Article 9, paragraph 1]

or implicit in which case a concept is used in a rule without any prior (or trailing) explicit definition.

Classifications are legal statements relating concepts and legal statements (through references) to each other. Classifications can state rules that *may* be followed as well as rules that *must* be followed. Often classifications take the form *x should/shall be treated as y*, which is the case in paragraph 1 of article 15 in the directive which reads:

> Electricity, gas, heat, refrigeration and the like shall be treated as tangible property.
> [5, Article 15, paragraph 1]

Classifications can also be implicit. This can happen in the situation where common sense determines how concepts are related to each other. An example is that the concepts vehicle and vessel are assumed to be non-overlapping[3].

Workflows are legal statements describing relative or absolute timing of events. They can be seen as a subcategory of classifications, but they deserve special attention because we believe they will be more challenging to model than non workflow classifications. An example is paragraph 3 of article 17 in[5] which reads:

[2] Unpublished joint work with Frantisek Sudzina (Copenhagen Business School), Ken Friis Larsen and Jakob Grue Simonsen (Department of Computer Science, University of Copenhagen).

[3] We disregard amphibious crafts (somehow this is always important to mention when you write for computer scientists).

> If one of the conditions governing eligibility under paragraph 2 is no longer met, the goods shall be regarded as having been transferred to another Member State. In such cases, the transfer shall be deemed to take place at the time when that condition ceases to be met.
>
> [5, Article 17, paragraph 3]

Clarifications are legal statements clarifying the meaning of definitions and classifications. As an example consider article 25 in the directive which reads:

> A supply of services may consist, inter alia, in one of the following transactions:
> (a) the assignment of intangible property, whether or not the subject of a document establishing title;
> (b) the obligation to refrain from an act, or to tolerate an act or situation;
> (c) the performance of services in pursuance of an order made by or in the name of a public authority or in pursuance of the law.
>
> [5, Article 25]

Legal statements that fall outside these categories can be classified as well, but they are not interesting with respect to the goals listed in the introduction. As an example of such a category we mention rules governing *necessary measures and regulatory behaviour*. Amongst others this category contains articles 19, 23 and 34.4[4] of the directive.

It is possible to classify legal statements in other ways as well. We have arrived at the scheme above through an analysis of the directive, where we had formal modelling in mind. From a logical modelling perspective the idea is that definitions provide the atoms with which rules can be built, while the other three categories contain the rules on which reasoning must be based.

2.3 Legal Doctrine

Legal Doctrine contains a range of meta principles that are used to disambiguate in situations where legal rules are in conflict. Most important is *Lex Superior*, which states that *superior norms suppress inferior norms.* If applicable this principle must always be used. The principle assumes that legal documents are arranged in a hierarchy where the most important document (e.g., the constitution) sits on the top. In the Danish system legal documents on the second level are called acts or statutes (in Danish: love), on the third level we have executive orders (in Danish: bekendtgørelser) and finally on the fourth level we have circulars (in Danish: cirkulærer).

[4] Article 34.4 is a shorthand notation for paragraph 4 of Article 34.

If two rules are in conflict and they belong to legal documents (possibly the same document) on the same level in the hierarchy we cannot use Lex Superior to disambiguate. In such situations the instructive principles *Lex Specialis* (particular norms suppress general norms) and *Lex Posterior* (later norms suppress earlier norms) can be used to guide the decision, but they can by no means be applied blindfoldedly without taking other issues such as the legislators' intent, the literal and teleological interpretation (Danish: ordlyds- og formålsfortolkning) etc. into account. The situation is the same if Lex Specialis and Lex Posterior contradict each other [1].

The principles Lex Superior, Lex Specialis and Lex Posterior are the so-called *first order rules of collision*. Second and higher order rules exist as well, but in practice it is only the second order rule stating that Lex Superior must be used (instead of Lex Specialis and Lex Posterior) if applicable that is used. Even so use of this rule is extremely infrequent, and the legal experts we have consulted with only knew of one recent case in Denmark[5] where this second order principle was applied.

Example 2.1. As an example of the use of Lex Specialis and Lex Superior consider the following example. Suppose the following rules exist:

1. The standard VAT rate applies to groceries.
2. The zero VAT rate applies to organic fruit and vegetables.

and suppose you are buying a punnet of organic strawberries. Clearly both rules apply, but they are in conflict! This is where Lex Specialis comes in. Because *organic fruit and vegetables* is more specific than the general term *groceries* the second rule must be used.

Now suppose the first rule is superior to (more important than) the second one. Then we must use the conclusion of the first rule since Lex Specialis can only be used if the rules are equally important.

2.3.1 Interpreting Exceptions

So far we have only covered the case where (main) rules are in conflict. An extra dimension is added when we consider exceptions to legal rules since exceptions can be nested and overlapping. In this section we analyse the situation. We begin with a presentation of the foundations and continue to present only the interesting cases (because of the large number of combinations of overlaps and nesting). To this end consider the rules[6] and exceptions in Figure 2.1.

[5] The Tvind verdict of 1998. The parliament had passed an act stating that Tvind schools were not eligible for government subsidies. However this act violated §3 in the constitution, tripartition of power, since the parliament had acted as executive as well as judicial power. In conclusion the Tvind verdict was ammended.

[6] We have found that (relevant) rules have the form $L \rightarrow R$. More about this later.

#	Rule
1	: $L_1 \rightarrow R_1$
2	: $L_2 \rightarrow R_2$
3	: **exception_to** $\{1\} : L_3 \rightarrow R_3$
4	: **exception_to** $\{1\} : L_4 \rightarrow R_4$
5	: **exception_to** $\{1, 3\} : L_5 \rightarrow R_5$
6	: **exception_to** $\{3, 4\} : L_6 \rightarrow R_6$

$$\begin{tikzcd}
 & & 1 & & 2 \\
 & 3 \arrow[ur, dashed] & & 4 \arrow[ul, dashed] & \\
5 \arrow[uurr, dashed, bend left] \arrow[ur, dashed] & & 6 \arrow[ul, dashed] \arrow[ur, dashed] & &
\end{tikzcd}$$

Fig. 2.1. On the left we have rules and exceptions. The *is exception to* graph on the right has an arc $A \dashrightarrow B$ iff A is an exception to B.

We think of a left hand side L_i of a rule i as the subset of the universe (of interpretation) consisting of elements that satisfy the conditions in L_i. An exception e with left hand side L_e to the rule i is a partitioning of the subset corresponding to L_i into the set where both L_i and L_e are true and the set where L_i is true and L_e is false. Likewise an exception with left hand side $L_{e'}$ to the previous exception is a partitioning of the set where L_i and L_e are true into the sets where additionally either $L_{e'}$ is true or false. A consequence of this interpretation is that we cannot apply the conclusions of exception e if the conditions in L_i are not satisfied. These observations are summarized graphically in Figure 2.2. Having the basics in place we continue with three

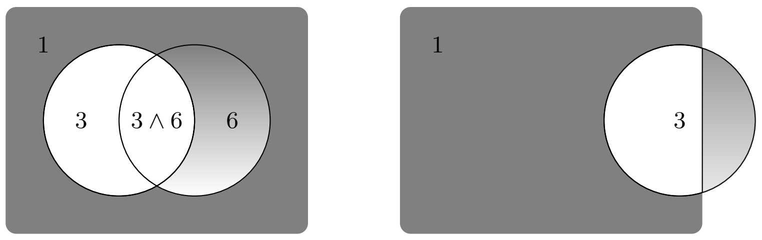


Fig. 2.2. The numbers reference the identifiers in Figure 2.1. On the left 3 is an exception to 1 and 6 is an exception to 3. Thus in area 3 we conclude R_3 and in area $3 \wedge 6$ we conclude R_6. In the shaded area however we conclude R_1 since 6 is not a direct exception to 1. On the right we cannot conclude anything in the shaded area because it is outside of 1 to which 3 is an exception.

nontrivial cases. The first two are treated in Figure 2.3 and the remaining one in Figure 2.4.

Area $3 \wedge 4$ in the left part of Figure 2.3 as well as areas $1 \wedge 2$ and $1 \wedge 2 \wedge 3$ of Figure 2.4 are interesting because we can ask what happens if R_3, R_4 respectively R_1, R_2 and R_2, R_3 are in conflict. These cases are similar yet different in the sense that the first case is a conflict between two exceptions, the second case is a conflict between two rules while the third case is a conflict between a rule and an exception (to another rule).

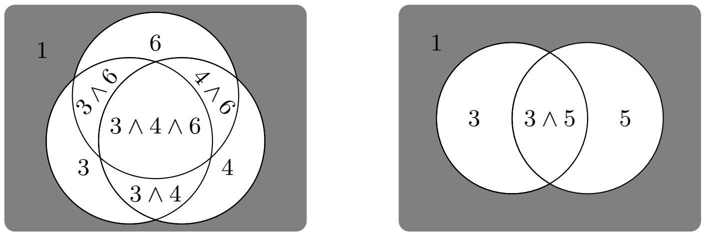


Fig. 2.3. The numbers reference the identifiers in Figure 2.1. On the left 3 and 4 are exceptions to 1. 6 is an exception to 3 and 4. In areas 3 and 4 we conclude R_3 and R_4 respectively. In areas $3 \wedge 6$, $4 \wedge 6$ and $3 \wedge 4 \wedge 6$ we conclude R_6. In area 6 we conclude R_1. Finally in area $3 \wedge 4$ the conclusion is more involved because R_3, R_4 can be in conflict. On the right 3 is an exception to 1, while 5 is an exception to both 1 and 3. In area 3 we conclude R_3, while in areas $3 \wedge 5$ and 5 we conclude R_5.

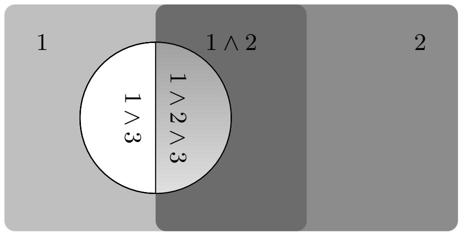


Fig. 2.4. The numbers reference the identifiers in Figure 2.1. 1 and 2 are overlapping rules and 3 is an exception to 1. In area 1 and 2 we conclude R_1 and R_2 respectively, while area $1 \wedge 3$ leads to the conclusion R_3. The conclusions in areas $1 \wedge 2$ and $1 \wedge 2 \wedge 3$ however is more involved, because R_1, R_2 respectively R_2, R_3 might be in conflict.

After having consulted with lawyers on how to resolve these conflicts, we found out that Lex Superior must be applied first of all. If this cannot resolve conflicts we must give precedence to main rules over exceptions (to other main rules). If this does not work either, only then do we continue to apply the instructive principles Lex Specialis and Lex Posterior. In Figure 2.5 we have summarized our observations in pseudo code even though a lot of details need to be filled in before we can call it an algorithm.

2.4 Summary

We have seen that legal documents carry structure on several levels. The purpose of the structure is to group related information in a way that is meaningful to humans and to provide means for legal statements to reference each other. Furthermore we have observed that (main) rules can be separated from their exceptions. We have also suggested a simple classification scheme,

```
DisambiguateRules($\mathcal{R}$ : <Set of rules>)
(* returns (Res,b) where b is true iff instructive
disambiguation has been used *)
if CollisionsIn($\mathcal{R}$)
  then $\mathcal{R}_1 \leftarrow$ PruneSuperior($\mathcal{R}$)
       if CollisionsIn($\mathcal{R}_1$)
         then $\mathcal{R}_2 \leftarrow$ PruneRulesVsExceptions($\mathcal{R}_1$)
              if CollisionsIn($\mathcal{R}_2$)
                then $G \leftarrow$ CreateLexSpecPostPrecedenceGraph($\mathcal{R}_2$)
                     (* G has an edge A < B iff B takes precedence
                     over A due to Lex Specialis or Lex Posterior *)
                     if NoCycles($G$)
                       then return (MaxElements($G$), $true$)
                       else raise Fail "Inconsistent conclusion"
                else return ($\mathcal{R}_2$, $false$)
         else return ($\mathcal{R}_1$, $false$)
  else return ($\mathcal{R}$, $false$)
```

Fig. 2.5. Pseudo algorithm for disambiguation of sets of legal rules.

which we believe can help to develop a modelling methodology part of which is to identify and capture the rules relevant in an ERP context.

Finally we have covered a range of legal meta principles and established a pseudo algorithm for disambiguation of sets of conflicting rules.

3

Support for Legal Rules in ERP Systems

In this chapter we investigate how VAT is handled from a user/partner perspective in NAV and SAP. We shall look at how the product that ships from the software vendor (Microsoft and SAP AG) can be configured to suit the customers needs.

NAV and SAP are humongous software systems. In the case of NAV, where we have had access to the source code, it has not been feasible to do a code analysis to uncover VAT functionality because VAT code is spread throughout the entire code base and attached to various NAV objects[1] such as *tables*, *forms* and *codeunits*. In the case of SAP we have been fortunate to have access to a SAP consultant and therefore we have based our account of the handling of VAT in SAP on interviews with him instead of experimenting with the system ourselves.

This chapter assumes a basic understanding of rudimentary accounting principles and terminology most notably double entry bookkeeping [8, 23].

3.1 Microsoft Dynamics NAV

In this section we explain how VAT is handled in Microsoft Dynamics NAV from a user perspective.

NAV targets Small and Medium sized Enterprises (SMEs) and is used in many countries and by many industries with different needs. Out of the box NAV (W1) supports most major VAT schemes. Where support is not built-in Microsoft relies on partners to *customise* NAV in order to provide the missing functionality. The partner strategy does not only apply to VAT, but is more or less the philosophy behind the development and maintenance

[1] Seven fixed (new ones cannot be added) object types exist in NAV. They are: tables, forms, reports, dataports, XMLports, codeunits and menusuites. All objects can be accessed through the *Object Designer*, which is available by pressing Shift ⇑ + F12 in the NAV client.

of NAV. Whether Microsoft decides to outsource development of specialised solutions to partners or to keep development in house is a question of market strategy, development costs, risks etc. which we shall not consider here. In what follows we describe the VAT support built into NAV (W1).

When you set up a company from scratch in NAV, you have the option to use a business template which provides standard configurations (which you can change later), amongst others for VAT. Here we are going to consider what lie below those templates. We begin by explaining what a quote (and an order) looks like in NAV.

3.1.1 Quotes and orders

A quote (and an order) consists of customer specific data having to do with invoicing, shipping etc. and a list of *sales lines* which make up the quote (or order). In NAV each customer belongs to exactly[2] one VAT Business Posting Group and this information is copied to the quote (or order) when it is created. Similarly each item (product) belongs to exactly one VAT Product Posting Group which is part of the information that is specified on the lines (a line describes the terms under which a quantity of a particular item is going to be sold). VAT is calculated per line and it is the combination of (the per quote/order) VAT Business Posting Group and (the per line) VAT Product Posting Group that determines how VAT is handled.

The handling of VAT belongs to the *Financial Management* module and is set up in the Posting Setup matrix[3] under Setup▷VAT Posting Group. The Posting Setup matrix is a way to specify a partial function from the two finite domains, VAT Business Posting Group and VAT Product Posting Group (which are also defined under Setup▷VAT Posting Group) into an 11-tuple of values that control the handling of VAT. Not all 11 values are needed in all cases and need only be specified if they are needed. Below we give a short description of each of the 11 values based on the Microsoft Dynamics NAV help utility:

VAT Identifier

The VAT Identifier can be used to group similar entries in the Posting Setup. Supposedly it is only descriptive and without semantic significance, but rules exist on how groups can be formed (e.g., it is not possible to use the same VAT Identifier on two different rows in the Posting Setup if they have the same VAT Prod. Posting Group, but different VAT %).

[2] Actually *at most* is more precise as NAV only requires you to fill in information if it is needed (which is checked at runtime).

[3] Not a matrix in the mathematical sense. The term comes from business vernacular and is widely (even though wrongly) used to describe any kind of data presented in rows and columns. Hence we shall adopt this denomination here too.

VAT %

The VAT % holds the VAT percentage for the particular combination of VAT Business Posting Group and VAT Product Posting Group.

VAT Calculation Type

The VAT Calculation Type controls how purchase and sales VAT is calculated for the combination of VAT Business Posting Group and VAT Product Posting Group. The following four schemes exist:

Normal VAT. This choice tells the system to use its own VAT calculation routines.

Reverse Charge VAT. This choice should be used when it is the obligation of the purchaser to settle VAT accounts with the tax authorities. Selecting this option will disable calculation and withholding of VAT on sales. On purchases it will debit the purchase VAT account and credit the reverse charge VAT account. This way it will not effect the company's financial statements.

Full VAT. This choice should be used if you want to post an entry consisting of VAT only. An important use is if you want to post a corrective entry in order to compensate for a prior VAT calculation error.

Sales Tax. This choice enables US sales tax instead of VAT.

Unrealized VAT

Unrealized VAT is VAT that is calculated, but first due when the invoice is paid. This option determines how a payment is allocated to the invoice amount (excluding VAT), to the VAT amount itself and how VAT amounts are transferred from the unrealized VAT account to the (realized) VAT account. In order to use this option unrealized VAT must be enabled (check mark) in the General Ledger Setup. The following schemes exist:

<Blank> means that unrealized VAT functionality is disabled.

Percentage means that each payment will cover VAT and invoice amount in proportion to the payment's percentage of the remaining invoice amount[4]. Paid VAT is transferred from the unrealized VAT account to the VAT account.

First means that a payment is allocated to VAT before the invoice amount. Paid VAT is transferred from the unrealized VAT account to the VAT account until the full VAT amount has been payed (at which point further payments will be allocated to the invoice amount).

[4] Exactly what this means is unclear.

Last Opposite of the First scheme. Payments will be allocated to the invoice amount (excluding VAT) before being allocated to the VAT account. Transfer from the unrealized VAT account to the VAT account will begin (corresponding to allocated payments) only when the full invoice amount has been paid.

First (Fully Paid) Similar to the First scheme, but paid VAT will be transferred from the unrealized VAT account to the VAT account only when the full VAT amount has been paid.

Last (Fully Paid) Similar to the Last scheme, but paid VAT will be transferred from the unrealized VAT account to the VAT account only when the full VAT amount has been paid.

Adjust for Payment Discount

Adjust for Payment Discount controls (check mark) whether VAT should be recalculated when payments that trigger payment discounts are posted. The sales VAT account will be updated accordingly. This feature must also be enabled (check mark) in the General Ledger Setup.

Sales VAT Account

Sales VAT Account controls what G/L account sales VAT should be posted to for the combination of VAT Business Posting Group and VAT Product Posting Group.

Sales VAT Unrealized Account

Sales VAT Unrealized Account controls what G/L account unrealized sales VAT should be posted to for the combination of VAT Business Posting Group and VAT Product Posting Group. Postings will be moved from this account to the sales VAT account according to the unrealized VAT scheme (see above) when payments arrive.

Purchase VAT Account

Purchase VAT Account controls what G/L account purchase VAT should be posted to for the combination of VAT Business Posting Group and VAT Product Posting Group.

Purchase VAT Unrealized Account

Purchase VAT Unrealized Account controls what G/L account unrealized purchase VAT should be posted to for the combination of VAT Business Posting Group and VAT Product Posting Group. Postings will be moved from this account to the purchase VAT account according to the unrealized VAT scheme (see above) when payments are posted.

Reverse Charge VAT Account

Reverse Charge VAT Account controls what G/L account reverse charge (purchase) VAT will be posted to (see VAT Calculation Type, Reverse Charge VAT) for the combination of VAT Business Posting Group and VAT Product Posting Group.

Reverse Charge Unrealized VAT Account

Reverse Charge Unrealized VAT Account controls what G/L account unrealized reverse charge (purchase) VAT will be posted to for the combination of VAT Business Posting Group and VAT Product Posting Group. On purchases this account will be debited and the reverse charge VAT account will be credited. When payments are posted they will trigger a corresponding move from this account to the purchase VAT account according to the unrealized VAT scheme.

3.1.2 Example

As a simple illustrative example let us consider the following:

A Danish vendor is selling a laptop to a Danish company.

We shall assume that the vendor has configured NAV using the Posting Setup shown in Figure 5.3 on page 46. Furthermore since the customer is a Danish company its VAT Business Posting Group will be `National` and since there are no exceptional rules with respect to VAT on laptops the full rate (25%) applies. Thus the VAT Product Posting Group of the item (laptop) has the value `VAT25`.

When the vendor processes the order in NAV the system will use the VAT Business Posting Group and the VAT Product Posting Group as a primary key in the Posting Setup. We see that the last line in Figure 5.3 matches and conclude that the VAT rate to apply is `25` (fourth column) and that the VAT Calculation Type is `Normal VAT`. In addition the Posting Setup specifies the account number we have to post sales VAT to (account `5610`).

3.1.3 Summing up

Microsoft Dynamics NAV supports a range of different VAT schemes. For a given sales line the VAT scheme to use is determined based on the VAT Business Posting Group and the VAT Product Posting Group. We observe that all schemes follow an implicit *quote to payment flow*, see Figure 3.1. A box in the figure indicates an event, which *might* trigger an action such as a debit of one account and a corresponding credit of another. A concrete VAT scheme is then a full specification of what actions are triggered by what events.

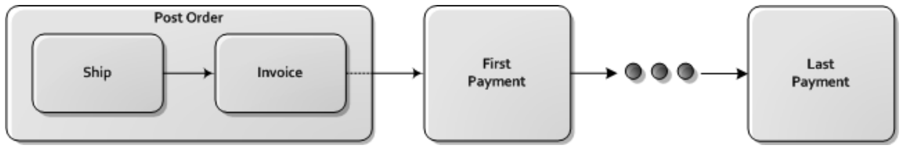


Fig. 3.1. Quote to Payment.

3.2 SAP

In this section we explain how VAT is handled in SAP (SAP ERP Core Version 6.0), an ERP system targeting mainly large businesses. Our description is based on a series of e-mail interviews with Kim Graversen, senior SAP consultant with Applicon A/S[5], who is specialised in the accounting modules within the SAP ERP Core environment. The e-mails (stripped from personal comments and small talk) are included in Appendix C.

As was the case with Microsoft Dynamics NAV we shall give an account from a user perspective. In SAP *Tax Codes* are used to identify VAT schemes (uniquely). We can think of a tax code as a name of the posting routine performing the actual calculations. In what follows we explain how the tax code (and thus VAT scheme) is determined for a given sales line[6] and we explain how VAT schemes are configured.

3.2.1 Finding the Tax Code

Figure 3.2 provides a simplified view of a sales order and related master data in SAP. The Tax Code that applies to a given sales line (on a sales order) is determined based on master data about the vendor (*country code*), item (*item VAT indicator* and *additional indicators*) and customer (*country code*, *customer VAT indicator* and *additional indicators*). With respect to customer data SAP distinguishes between *sold-to*, *ship-to*, *bill-to* and *payer* relationships (compare this to NAV where a sales order is related to one set of customer data only). The last field shown on the simplified sales order in Figure 3.2 is *sales-org*, which is used to determine the *chart of accounts* the sales order should be posted to[7] (NAV only supports a single chart of accounts).

Figure 3.3 illustrates how the tax code is found for a given sales line. This is done by running the *access sequence*, which is a prioritised two stage search for matching conditions. In the first stage matching is done on the country code of the vendor (the location from where the supply is sent) and the country codes of the customer (found via the *sold-to*, *ship-to* etc. relations). The goal

[5] Applicon is a Danish SAP consultancy.

[6] Tax can be calculated pr. sales line or any other object that eventually contains sales lines such as sales orders, bookkeeping entries or profit centres.

[7] In SAP a sales-org has a many - one relationship with *company-code*, which in turn has a many - one relationship with chart of accounts.

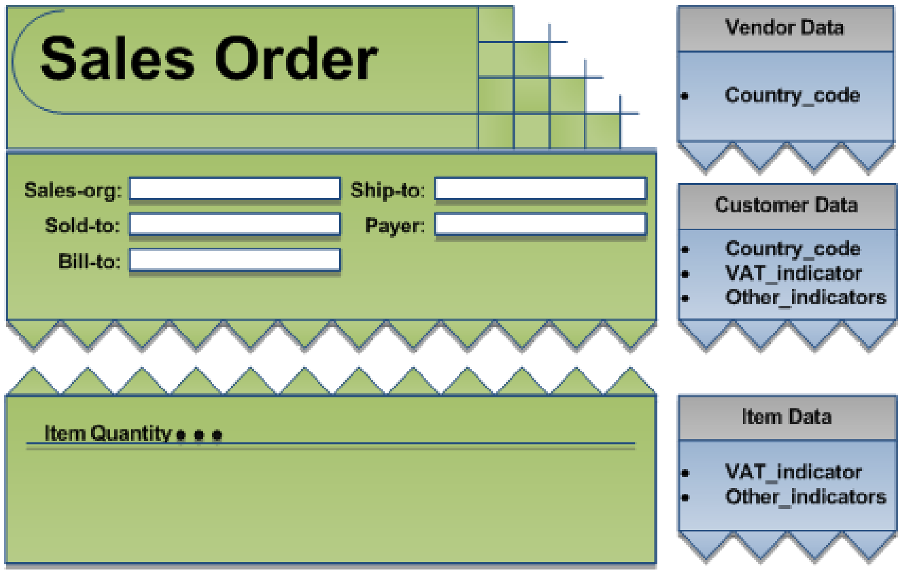


Fig. 3.2. Simplified view of a sales order and related master data in SAP.

of the first stage is to determine whether it is a domestic or an export sale and to identify a table containing additional conditions if any (e.g., on master data such as item and customer VAT indicators) to match with in the second stage. The goal of the second stage is to look up the tax code in the table identified in stage one based on a primary key consisting of data derived from the sales line.

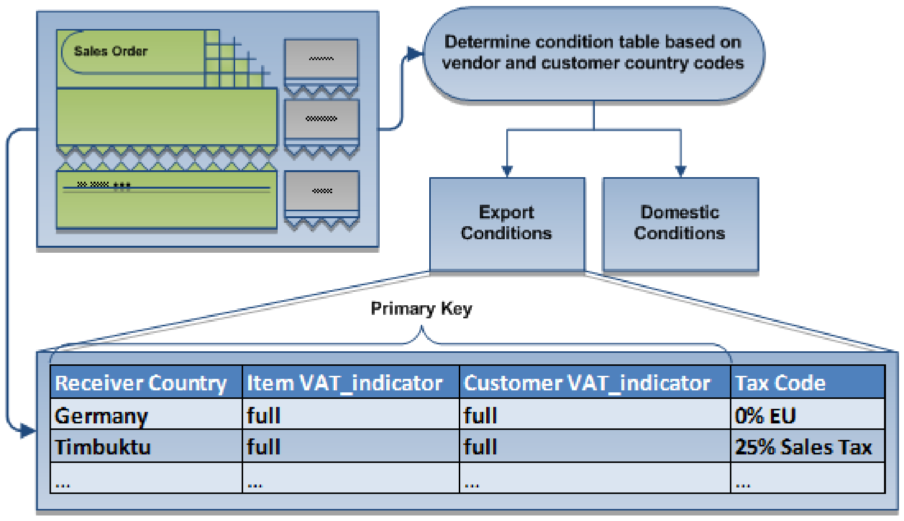
Example

Let us consider the example from before:

> A Danish vendor is selling a laptop to a Danish company.

In the case of SAP this means that the vendor country code determined via the *sales-org* field and the customer country codes determined via the *sold-to*, *ship-to*, *bill-to* and *payer* fields are all `Denmark`. Furthermore the customer and item VAT identifiers are `full`.

In order to determine the tax code (primary key controlling how to handle VAT) the system will run the access sequence. In is not entirely clear to us which of the country codes of the customer the system will use in stage one (it makes sense to use the one determined via the *sold-to* relationship), but the important thing is that the outcome will identify a table containing additional conditions on domestic sales in Denmark. That table is similar to the one depicted in Figure 3.3, but without the column *Receiver Country* (domestic sale implies that the receiver country is Denmark). Using the customer and


Sales Order
Determine condition table based on vendor and customer country codes
Export Conditions
Domestic Conditions
Primary Key

Receiver Country	Item VAT_indicator	Customer VAT_indicator	Tax Code
Germany	full	full	0% EU
Timbuktu	full	full	25% Sales Tax
...	...	...	...

Fig. 3.3. Finding the Tax Code.

item VAT identifiers from the sales order as a primary key in this table we can determine the tax code to use, say `S1`. *To be continued....*

3.2.2 How to set up VAT Schemes

Having explained how a tax code is determined for a given sales line, it is time to describe how the VAT scheme identified by a tax code is set up. We do this based on SAP documentation [16], and on interviews with Kim Graversen.

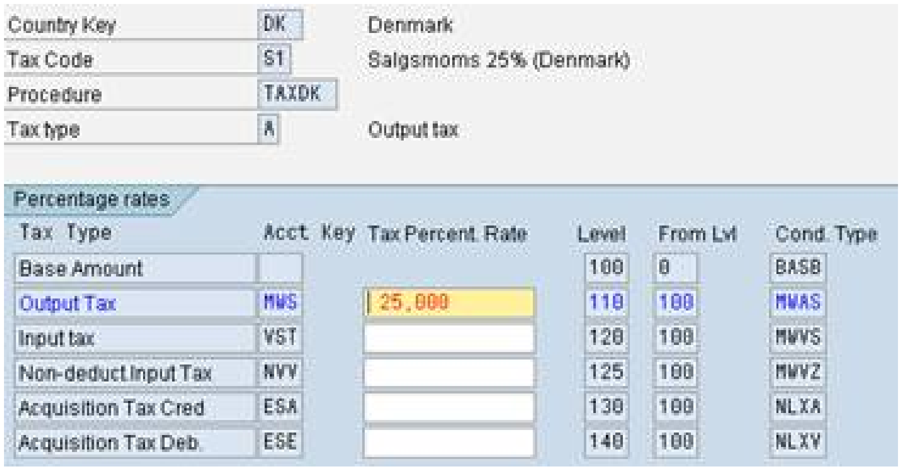
SAP tries to support good reuse of code and therefore the individual VAT schemes that SAP supports have been broken down into atomic tax posting routines, called *Tax Types*, parametrised over percentage and G/L accounts (as well as some additional details, which we omit here). Tax types can be created semi automatically based on answers to a set of questions[8] such as what *calculation type* (percentage included or percentage separate) to use, but for the purpose of this description we shall omit any further details on that.

Out of the box SAP comes with a lot of predefined tax types from which country specific VAT schemes, represented by tax codes, are implemented. As it is common practise, we shall use the term tax code to also refer to the VAT Scheme it identifies as well.

For each country code SAP provides a *Calculation Procedure*[9], which specifies the details needed in order to create tax codes from tax types. In Figure 3.4 we show the set up of tax code `S1` in the calculation procedure for `DK`. In the

[8] See: `http://help.sap.com/saphelp_erp60_sp/helpdata/en/e5/077e394acd11d182b90000e829fbfe/content.htm`.

[9] In rare cases such as ex Yugoslavia and ex Soviet countries you might need to define the Calculation Procedure yourself.



Country Key	DK	Denmark
Tax Code	S1	Salgsmoms 25% (Denmark)
Procedure	TAXDK	
Tax type	A	Output tax

Percentage rates

Tax Type	Acct Key	Tax Percent. Rate	Level	From Lvl	Cond. Type
Base Amount			100	0	BASB
Output Tax	MWS	25,000	110	100	MWAS
Input tax	VST		120	100	MWVS
Non-deduct.Input Tax	NVV		125	100	MWVZ
Acquisition Tax Cred	ESA		130	100	NLXA
Acquisition Tax Deb.	ESE		140	100	NLXV

Fig. 3.4. Set up of Tax Code `S1` in the calculation procedure for `DK`.

top part of the figure the Country Key `DK` led to the identification of the Procedure `TAXDK` of, which `S1` is a tax code. First of all `S1` has *Tax type* `A`, which means it is an output tax[10]. The only other option for tax type is `V` indicating input tax. The lower part contains 6 lines. The first is the base amount from which taxes must be calculated and the remaining 5 lines correspond to the (subset of all) tax types that are relevant for the calculation procedure. In what follows we explain what the columns in the lower part mean:

Tax Type determines the tax type with related posting routines etc.
Acct Key is used in connection with the chart of accounts id (derived from the sales-org field on the sales order) and optionally the tax code to determine the G/L account to post to. See Figure 3.5.
Tax Percent Rate is the tax rate to apply.
Level is a numeric id for the line/row.
From Lvl determines what line to use as input amount. In more sophisticated tax schemes it might be the case that you calculate an extra tax based on the result of an initial tax calculation (which in turn is based on the base amount).
Cond Type is used to identify a *validation table* to use for the tax type in question. This allows for extra flexibility in the configuration of tax codes, but according to Kim Graversen the defaults (the ones shown) are always used in practise.

[10] Unfortunately the translation of SAP from German into English has resulted in two different concepts being named *tax type*. However, it should be clear from the context which one is meant.

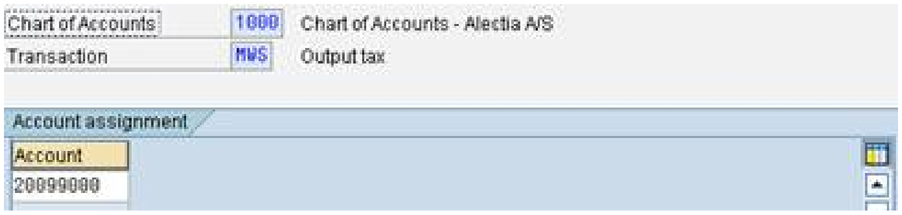


Chart of Accounts	1000	Chart of Accounts - Alectia A/S
Transaction	MWS	Output tax

Account assignment

Account
20099000

Fig. 3.5. Account assignment for Acct Key `MWS` and Chart of Accounts id `1000`.

Example Continued

Previously we determined that tax code `S1` applies to our sale (of a laptop). Figure 3.4 shows how the VAT scheme identified by tax code `S1` is set up. In the lower part of the figure the first line/row represents the *Base Amount* in this case the sales price without VAT. The remaining lines specify how to calculate different kinds of tax. In all cases except *Output Tax* (VAT) the *Tax Percent Rate* is empty (0) meaning that tax code `S1` only applies a single posting routine (tax type) namely *Output Tax*. As mentioned tax types are atomic posting routines parametrized over percentage (column three) and the G/L account to post to. The G/L account is determined from the value in column two `MWS` together with the chart of accounts id (determined via the *sales-org* field on the sales order), see Figure 3.5.

Finally the column *From Lvl* tells us that the calculation of output tax should take as its input the outcome of the calculation made by line `100` in other words output tax is `25.000` percent of the sales price without VAT (the base amount).

3.2.3 Summing up

SAP supports many different VAT schemes, which are represented by tax codes. For most countries tax codes are built-in, but if not new ones can be constructed from (built-in or user defined) atomic posting routines called tax types. The tax code to use for a given sales line is determined based on a two stage access sequence, which takes a range of parameters such as the locations of the vendor and the customer (sold-to, ship-to etc.), VAT indicators etc. into account. An inspection of the SAP documentation [16] with respect to withholding tax[11] (the SAP term for unrealized VAT) reveals that SAP follows a quote to payment flow similar to NAV, see Figure 3.1.

[11] See: `http://help.sap.com/saphelp\_erp60\_sp/helpdata/en/2d/936b3ae886616ae10000000a114084/content.htm`.

3.3 Comparison and Summary

In this chapter we have investigated two major ERP systems with respect to VAT. We have learnt that both systems follow a quote to payment flow similar to that in Figure 3.1, but that they differ in the way VAT schemes are implemented and in the way they determine which VAT scheme to use. While Microsoft Dynamics NAV uses an approach based solely on the value of a customer and item identifier[12] SAP uses a more detailed two stage approach.

As expected SAP is more configurable and adaptable than Microsoft Dynamics NAV. However it comes as a surprise that implementation of new VAT schemes in SAP seems (we have no firsthand experience) to be rather easy compared to Microsoft Dynamics NAV due to the way VAT schemes are formulated in SAP using (predefined) tax types.

3.3.1 Lack of Transparency is the Problem

We have found that both systems lack what we refer to as *transparency* or *direct support* for legal rules, meaning that an *isomorphic*[13] mapping of legal statements to code and back is missing completely. Thus it is hard if not impossible in current systems to say what parts of the relevant legislation they implement. One view, which seems to fit well with our investigations, is to say that ERP systems support and implement the most general rules and leave out a lot of the exceptional situations and details.

The problem occurs when ERP systems evolve over time and the need for support for exceptional situations arise. This can be due to requirements imposed by legal authorities or by companies (it is often the case that exceptions to general rules list situations where less or no VAT is due). In either case changes such as add-ons and updates to Microsoft Dynamics NAV (we speculate the situation is similar for SAP) require the involvement of many people, programmers and legal domain experts alike. As described in the introduction this makes interaction with the VAT code time consuming, expensive, and error prone.

[12] Microsoft Dynamics NAV only supports one sales organisation pr. chart of accounts. Thus information about the vendor of a particular sale is implicitly known.
[13] A precise definition will be given later.

4

Requirements on a Model

As described in the introduction our goal is to allow domain experts to create models of legal rules, that can be used by a query engine to answer questions about VAT. We stated that such a modelling language should provide better support for the complexity (which stems from the large number of countries with slightly different VAT rules that needs to be supported) and argued that it could ultimately lead to lower development and maintenance cost. In this chapter we present the requirements we think our goal induces on a modelling language for VAT. The requirements will be based on our findings in Chapter 2 and 3 and on principles advocated in the literature as well as our experience from initial experiments. We begin with an introduction of an overall modelling principle. A summary of all requirements introduced below can be found in Figure 4.2.

4.1 The Isomorphism Principle

In Definition 4.1 we present the *isomorphism principle* [20] (following Karpf [1989]).

Definition 4.1 (Isomorphism principle). *We say that a representation of a legal system is an* isomorphism *if:*

1. *Each legal source is represented separately.*
2. *The representation preserves the structure of each legal source.*
3. *The representation preserves the traditional mutual relations, references and connections between the legal sources.*
4. *The representation of the legal sources and their mutual relations ... is separate from all other parts of the model, notably representation of queries and facts management.*

Karpf has a fifth requirement:

> If procedural law is part of the domain of the model then the law module will have representation of material as well as procedural rules and it is demanded that the whole system functions in accordance with and in the order following the procedural rules.
>
> [20]

As in [20] we find this requirement unreasonable. We believe that a legal model should be as declarative as possible and therefore it seems wrong to require that procedural rules, which we call workflow rules, guide the order in which the system (query engine) processes the rules. One can imagine that the reason behind this requirement is that the query engine should be able to provide evidence (e.g., inference trees) to support the conclusions it reaches. This is still a valid concern. In particular human dialogue (if necessary) should be structured and tied to a particular section of the model in such a way that it is clear how the system arrived at its conclusions.

Whether the isomorphism principle is a good design principle or not has been debated in the literature. The reason why we choose to adopt it here is because it guarantees *locality of change*, which is a very important property of any model that is updated frequently. Locality of change states that an update in the underlying legal source should trigger an update of similar size in the model. In other words it ensures that local changes in the legal source cannot trigger global changes in the model.

Furthermore the preservation of structure and mutual relations should help domain experts to navigate the model because they know the structure from the legal source already. All in all we believe that following the isomorphism principle leads to better support for complexity and to transparency between legal source and model.

We can now state the first requirement:

> The modelling language should support the isomorphism principle, in particular it should support the structure and referencing presented in Section 2.1 including separation of rules and their exceptions.

4.2 Tractable Analysis vs. Expressiveness

Clearly a modelling language for legal rules must be able to express the semantics of relevant rules in order to provide any value at all. However it is a classic fact in computer science that the more expressive a language is the less amenable it is to automated analysis. Therefore we want to create a language with sufficient, but otherwise minimal expressive power. In this section we present things that the language must be able to express.

4.2.1 Concepts

Legal concepts are the atoms from which legal statements are built. One of the reasons why legal vernacular can be deemed to be semi-formal is that concepts carry precise meanings and that they are used systematically (synonyms should be avoided). Furthermore concepts can have relations to each other such as the *is-a* (sub-concept) and *disjoint* relationships and such relationships can be implicit.

While implicit relationships (determined by common sense) work well in legal documents it is necessary to make them explicit in a formal model. This threatens to break the isomorphism principle and raises the question of where to put implicit assumptions about concept relationships. Furthermore the interpretation of implicit assumptions is left entirely to the modeller who does the job. Thus the correctness of implicit assumptions can be debatable. For these reasons it is desirable to be able to distinguish (e.g., using tags) implicit assumptions from regular ones in a model. Finally it is advisable to (re)state implicit assumptions in all the places where they are used, because then a deletion of an implicit assumption in one place cannot have unintended effects in other places. If this is done we retain locality of change, which was the main motivator behind the adoption of the isomorphism principle.

Sometimes concepts (e.g., *exempt supplies*) are defined by specifying members. This suggests that we can think of concepts as sets of individuals, which is exactly what is done in OWL (Web Ontology Language, [3]), which was the starting point for this work. In the examples we have encountered during our initial investigations it has only been necessary to stipulate membership of a finite number of individuals for each concept. While this does not rule out that concepts can be infinite it does hint that concepts are finite modulo observational equivalence of individuals with respect to the model. This observation is interesting with respect to implementation since it suggests that a concept can be represented via a finite number of propositional symbols in propositional logic.

Based on this we state the following requirement:

> The modelling language should support definition of legal concepts as well as specification of sub-concept and disjointness relationships. In addition it should be possible to state that a relationship between concepts is implicit.

4.2.2 Structure of Rules, Operations and Workflows

For the purpose at hand, described in Section 1.3, legal statements can be thought of as logical statements with the slight difference that some rules are optional (rules using *may* and *should* instead of *must* and *shall*, see Section 2.2). If rules can be optional, then so can inferences, which leads us to the problem of what to do if a mandatory as well as an optional inference of some fact exists (not to mention what to do, when they are in conflict). For simplicity we choose only to support rules that *must* be followed.

Some legal statements describe the timing of events; and actions, such as billing a customer, can be triggered by events. This suggests that the modelling language includes some kind of support for workflows. A legal model can be viewed as a set of rules restricting the number of ways you are allowed to perform certain obligatory actions. Therefore a procedural (executable) specification of workflows, such as BPEL, is not optimal and would most likely introduce (unnecessary) restrictions on the number of ways vendors can implement systems. Another option is to include declarative support for workflows. Van der Aalst and Pesic [22] have done this in a graphical language, which is mapped to *Linear Temporal Logic* (LTL). We shall adopt the latter approach and include declarative support for workflows, but for the moment not tie us to a particular kind of temporal logic.

We can now state the following requirement on the modelling language:

> The modeling language should support declarative specification of workflows based on modal logic. Furthermore the language need not support optional rules only mandatory ones.

Initial experiments have shown that rules take the form $(L_1 \wedge L_2 \wedge \cdots \wedge L_n) \rightarrow (R_1 \wedge R_2 \wedge \cdots \wedge R_m)$ and that both L and R expressions can contain comparisons for equality on Presburger Arithmetic expressions (including multiplication by constants) as well as comparisons for equality and tests for sub-concept relationships. In addition L expressions can contain inequalities on Presburger Arithmetic expressions, they can be negated and finally L respectively R expressions can make references to left, respectively right hand sides (conjunctions of L, respectively R expressions) of other legal statements as well.

In summary we require that:

> The logic underlying the modelling language should support rules of the form $(L_1 \wedge L_2 \wedge \cdots \wedge L_n) \rightarrow (R_1 \wedge R_2 \wedge \cdots \wedge R_m)$ where L and R expressions can contain comparisons for equality on Presburger Arithmetic expressions as well as comparisons for equality and tests for sub-concept relationships. In addition L expressions should support inequalities on Presburger Arithmetic expressions and negation. Finally left hands sides should be able to reference left hand sides of other legal statements and similarly for right hands sides.

4.3 Legal Reasoning and Collision Rules

We use the term *Legal Reasoning* and the phrase *reasoning in the legal domain* to mean the way legal practitioners apply the law (a set of rules) to situations (sets of facts) in the real world.

Reasoning in the legal domain applies an open world assumption meaning that the absence of knowledge cannot be used to conclude the opposite. Furthermore reasoning must be constructive. A priori we can construct two pseudo algorithms for legal reasoning. The pseudo algorithms are presented in Figure 4.1 and are almost identical. The difference is that ReasonNoCyc assumes that the conclusion (right hand side) of a rule (implication) does not depend on how its conditions (left hand side) were satisfied[1]. Therefore it suffices to apply a rule only once if at all and thus the pseudo algorithm need not consider rules already applied when it calls itself recursively (in line 6).

```
ReasonNoCyc($A$ : <Set of assumptions>, $\mathcal{R}$ : <Set of rules>, $istrDis_1$ : <boolean>)
1  $\mathcal{R}_1 \leftarrow$ FindApplicableRules($A, \mathcal{R}$)
2  $(\mathcal{R}_2, istrDis_2) \leftarrow$ DisambiguateRules($\mathcal{R}_1$)
3  $A' \leftarrow A \cup$ ConclusionsOf($\mathcal{R}_2$)
4  if $A' = A$
5     then return $(A, istrDis_1 \vee istrDis_2)$
6     else  ReasonNoCyc($A', \mathcal{R} \setminus \mathcal{R}_1, istrDis_1 \vee istrDis_2$)

ReasonCyc($A$ : <Set of assumptions>, $\mathcal{R}$ : <Set of rules>, $istrDis_1$ : <boolean>)
1  $\mathcal{R}_1 \leftarrow$ FindApplicableRules($A, \mathcal{R}$)
2  $(\mathcal{R}_2, istrDis_2) \leftarrow$ DisambiguateRules($\mathcal{R}_1$)
3  $A' \leftarrow A \cup$ ConclusionsOf'($\mathcal{R}_2, A$)
4  if $A' = A$
5     then return $(A, istrDis_1 \vee istrDis_2)$
6     else  ReasonCyc($A', \mathcal{R}, istrDis_1 \vee istrDis_2$)
```

Fig. 4.1. Pseudo algorithms for Legal Reasoning. The DisambiguateRules algorithm is defined in Figure 2.5. It is assumed that a fixed point will be reached eventually in line 4.

This leads to the following requirement:

> Reasoning about models should follow either ReasonNoCyc or ReasonCyc as appropriate and consequently conflicts between rules should be resolved using DisambiguateRules. Therefore the modelling language must support the first order collision rules Lex Superior, Lex Specialis and Lex Posterior.

4.4 Tool Support and Usage

It is not reasonable to expect that domain experts interact with an ASCII version of the modelling language directly (e.g., through a general purpose text editor). Therefore we envision that models are created and maintained through a GUI. While GUI design and development is outside the scope of this project it is important to understand that it is the combination of language and GUI features, which enable domain experts to do the modelling. One way to design a GUI is by using graphical notation as sugar for common

[1] In the code ConclusionOf(R_2) is used instead of ConclusionOf'(R_2, A).

constructions in the modelling language (e.g., similar to van der Aalst and Pesic [22]).

Another important issue is that modelling will most likely involve many people with different skills. Therefore the (combination of) modelling language and GUI should support good software engineering principles, including support for transfer of the model between persons with different areas of responsibility [2].

This leads to the following requirement:

> Tool support should be available, and the combination of tools and modelling language should support good software engineering principles in particular transfer of the model between persons with different areas of responsibility.

Finally we must not forget that the idea with the modelling language is to build models that, amongst others, can be queried and analysed. A query is a request to answer a question and to provide justification for the answer based on a possibly empty set of assumptions. Furthermore it is desirable that the query engine can guide this process by requesting the user to provide only the input necessary to answer the question(s) at hand. Since the input necessary to answer a query can depend on input already given we suggest to use a query protocol where the user inputs one assumption at a time based on a set of choices that is dynamically recomputed between every selection.

Thus we have the following critical requirement:

> It should be possible to query models built using the modelling language.

[2] The content in this section is heavily based on earlier work [13]. To some extend we use exactly the same wording.

Requirements

1. The modelling language should support the isomorphism principle, in particular it should support the structure and referencing presented in Section 2.1 including separation of rules and their exceptions.
2. Reasoning about models should follow either ReasonNoCyc or ReasonCyc as appropriate and consequently conflicts between rules should be resolved using DisambiguateRules. Therefore the modelling language must support the first order collision rules Lex Superior, Lex Specialis and Lex Posterior.
3. The modelling language should support definition of legal concepts as well as specification of sub-concept and disjointness relationships. In addition it should be possible to state that a relationship between concepts is implicit.
4. The modeling language should support declarative specification of workflows based on modal logic. Furthermore the language should not support optional rules only mandatory ones.
5. The logic underlying the modelling language should support rules of the form $(L_1 \wedge L_2 \wedge \cdots \wedge L_n) \rightarrow (R_1 \wedge R_2 \wedge \cdots \wedge R_m)$ where L and R expressions can contain comparisons for equality on Presburger Arithmetic expressions as well as comparisons for equality and tests for sub-concept relationships. In addition L expressions should support inequalities on Presburger Arithmetic expressions and negation. Finally left hands sides should be able to reference left hand sides of other legal statements and similarly for right hands sides.
6. Tool support should be available, and the combination of tools and modelling language should support good software engineering principles in particular transfer of the model between persons with different areas of responsibility.
7. It should be possible to query models built using the modelling language.

Fig. 4.2. Requirements on a Modelling Language for Legal Rules.

Part II

LLMBRR

5 Discussion

The analysis in part I resulted in a list of requirements, including two pseudo algorithms describing the principles in legal reasoning. We used the term *pseudo* to indicate that the algorithms are not implementable as is. The reason is that we have not got a formal understanding of the concept of rules yet, and consequently we cannot explain (formally) what it means that a rule is *applicable*, what a *conclusion* of a rule is, and finally what it means that (a set of) rules are *in conflict*[1].

A formal understanding of rules can be approached from several angles. Ours has been to write down a syntax of a DSL for legal modeling, which we call LLMBRR[2], and trying to explain informally what the language constructs mean and how legal reasoning is performed (operationally from a domain experts point of view) in the context of this language.

Having and informal understanding of the language will enable us to research and compare how well suited different logics for knowledge representation are with respect to (legal) rule modeling and ultimately with respect to giving semantics to our language. Logics of interest include, but are not limited to, description logics, modal logics (e.g., deontic and hybrid logic), and restricted versions of FOL (e.g., Horn clause and linear logic).

One of the benefits of having a syntax from early on (as opposed to designing the syntax from the semantics once established) is that we can experiment with modeling of (real world) legal rules in the language and this way gain insights about how the semantics ought to behave, and furthermore we can correct bad design with little effort before it becomes a problem. The downside of course is that it is hard to say anything concrete about a language without meaning.

[1] These are the notions underlying the sub-algorithms FindApplicableRules, ConclusionsOf and CollisionsIn used in the pseudo algorithms in part I on pages 18 and 35.

[2] An abbreviation of *Language for Logical Modeling of Business Rules and Regulations.*

In what follows we present and explain the newest version of the syntax of LLMBRR and we give examples several examples.

5.1 LLMBRR Syntax

LLMBRR is designed to meet the requirements in Figure 4.2. Its syntax is defined as follows:

Definition 5.1 (LLMBRR Syntax). *The syntax of core LLMBRR models is defined in Figure 5.1, where keywords have been highlighted using* ***bold-face****.*

The syntax uses several meta variables to range over various sets, they are:

`di`	*ranges over Doc*	*The set of Document Identifiers.*
`il`	*ranges over* $(\mathbb{N}_0, \leq)$	*The total order of importance levels.*
`lsi`	*ranges over Legalsec*	*The set of Legal Section Identifiers.*
`ai`	*ranges over Article*	*The set of Article Identifiers.*
`subi`	*ranges over Subarticle*	*The set of Subarticle Identifiers.*
`li`	*ranges over Lstmt*	*The set of Legal Statement Identifiers.*
`ci`	*ranges over Concept*	*The set of Concept Identifiers.*
`afid`	*ranges over Afun*	*The set of Aspect Function Identifiers.*
$\overline{\texttt{c}}$	*ranges over* $\overline{\mathbb{N}}_0$	*The set of Non negative Integer Symbols.*
`<date>`	*ranges over* $(Date, \leq_D)$	*The total order of dates.*
`<str>`	*ranges over String*	*The set of Strings.*

Remark 5.2. LLMBRR uses standard associativity and precedence rules. Thus **;**, **and**, $+$ and $*$ associates to the left and $*$ binds most tightly followed in order by $+$, **not**, **and** and **;**.

We have tried to keep the number of operators in core LLMBRR small in order to minimize the size of a formal semantics for the language. Using the operators of core LLMBRR we define sugar notations for common expressions. This is done in Figure 5.2.

5.1.1 Explaining the Syntax

LLMBRR models consist of a non-empty set of **legaldoc**s followed by an optional **aliases** section. Each legaldoc is intended to model a document or act such as the European Directive on VAT [5] so whenever a model contains information from multiple legal sources is should contain a corresponding number of legaldocs. Each legaldoc must have a unique name and consists of a **header** and a **body**. The header contains information about the date where the legal source modeled was **enacted** (to be used in connection with Lex Posterior) and the **importance-level** of the document (to be used in

```
Model       ::= model LD^1+ end
              | model LD^1+ Aliases end
LD          ::= legaldoc di Header Body
Header      ::= header
                  enacted: <date>
                  importance-level: il
Body        ::= body Legalsec^1+
Legalsec    ::= legalsec lsi Optname LSBody end
Optname     ::= · | <str>
LSBody      ::= Legalsec^1+ | Article^1+
Article     ::= article ai SABody
SABody      ::= Subarticle^1+ | Lstmt^1+
Subarticle  ::= subarticle subi SABody end
Lstmt       ::= lstmt li: LSTBody
LSTBody     ::= IDef | Def | exception_to {Lis} Rule | Rule
IDef        ::= implicit Def
Def         ::= all_disjoint {Cis} | Ci sub U{Cis} | Def; Def
Cis         ::= Ci | Ci, Cis
Ci          ::= di.ci | ci
Lis         ::= Li | Li, Lis
Li          ::= di.li | li
Rule        ::= LHS -> RHS
LHS         ::= Lexpr | lref {Lis} | LHS and LHS
RHS         ::= Rexpr | rref {Lis} | RHS and RHS
Lexpr       ::= not Lexpr | Presb < Presb | this in Ci | Afid(this) in Ci
Rexpr       ::= Presb = Presb | this in Ci | Afid(this) in Ci
Presb       ::= Afid(this) | c̄ | Presb + Presb
Afid        ::= di.afid | afid
Aliases     ::= aliases
                  concepts: ConceptAl
                  aspectfunctions: AFAl
ConceptAl   ::= · | di.ci sub di.ci, ConceptAl
AFAl        ::= · | di.afid eq di.afid, AFAl
```

Fig. 5.1. BNF grammar for core LLMBRR.

connection with Lex Superior). The body consists of a structured collection of definitions, exceptions and rules, which are uniquely identifiable. We have chosen not to over specify LLMBRR to the European Directive on VAT [5] by strictly following its structure. Instead we allow for more flexibility by way of general nesting of legal statements, since it might be the case that other legal documents use a slightly different structure. However, we have retained the concept of an **article** (which can be subdivided further) because this is a central concept in the legal domain.

Definitions, which can be tagged with the **implicit** keyword, state relationships between legal concepts, which we can think of as sets. Exceptions

$$
\begin{array}{lll}
\mathit{Def}\ : & & \\
 & \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\}\ \mathbf{sub}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_m\} & \equiv_{\mathtt{stx}}\ Ci_1\ \mathbf{sub}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_m\};\\
 & & \phantom{\equiv_{\mathtt{stx}}}\ Ci_2\ \mathbf{sub}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_m\};\\
 & & \phantom{\equiv_{\mathtt{stx}}}\ \ldots\\
 & & \phantom{\equiv_{\mathtt{stx}}}\ Ci_n\ \mathbf{sub}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_m\}\\
 & Ci\ \mathbf{sub}\ Ci' & \equiv_{\mathtt{stx}}\ Ci\ \mathbf{sub}\ \mathbf{U}\{Ci'\}\\
 & \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\}\ \mathbf{sub}\ Ci' & \equiv_{\mathtt{stx}}\ \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\}\ \mathbf{sub}\ \mathbf{U}\{Ci'\}\\
 & & \\
 & \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\} = \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_n\} & \equiv_{\mathtt{stx}}\ \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\}\ \mathbf{sub}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_n\};\\
 & & \phantom{\equiv_{\mathtt{stx}}}\ \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_n\}\ \mathbf{sub}\ \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\}\\
 & Ci = Ci' & \equiv_{\mathtt{stx}}\ \mathbf{U}\{Ci\} = \mathbf{U}\{Ci'\}\\
 & Ci = \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_n\} & \equiv_{\mathtt{stx}}\ \mathbf{U}\{Ci\} = \mathbf{U}\{Ci'_1, Ci'_2, \ldots, Ci'_n\}\\
 & \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\} = Ci' & \equiv_{\mathtt{stx}}\ \mathbf{U}\{Ci_1, Ci_2, \ldots, Ci_n\} = \mathbf{U}\{Ci'\}\\
 & & \\
\mathit{Lexpr}\ : & & \\
 & \mathit{Presb}_1 > \mathit{Presb}_2 & \equiv_{\mathtt{stx}}\ \mathit{Presb}_2 < \mathit{Presb}_1\\
 & \mathit{Presb}_1 >= \mathit{Presb}_2 & \equiv_{\mathtt{stx}}\ \mathbf{not}\,(\mathit{Presb}_1 < \mathit{Presb}_2)\\
 & \mathit{Presb}_1 <= \mathit{Presb}_2 & \equiv_{\mathtt{stx}}\ \mathit{Presb}_2 >= \mathit{Presb}_1\\
 & & \\
\mathit{LHS}\ : & & \\
 & \mathit{Presb}_1 = \mathit{Presb}_2 & \equiv_{\mathtt{stx}}\ \mathit{Presb}_1 <= \mathit{Presb}_2\ \mathbf{and}\ \mathit{Presb}_1 >= \mathit{Presb}_2\\
 & & \\
\mathit{Presb}\ : & & \\
 & \overline{\mathtt{c}}\ *\ \mathit{Presb} & \equiv_{\mathtt{stx}}\ \underbrace{\mathit{Presb} + \mathit{Presb} + \cdots + \mathit{Presb}}_{\mathtt{c}}
\end{array}
$$

Fig. 5.2. Extended BNF for LLMBRR.

are special rules that refine the meaning of other rules (or exceptions) (see Section 2.3.1), and rules are Horn clause like expressions built from references to other rules, (in)equalities on Presburger Arithmetic expressions and expressions stating membership of concepts. Finally the aliases section binds legaldocs together by stating relationships on concepts and aspect functions.

We note that concepts and aspect functions (explained below) do not need to be declared explicitly. The mere fact that they are used in legal statements declares them implicitly.

5.1.2 The Meaning of this

In this section we explain the intended meaning of the keyword **this** and its use in connection with aspect functions and *element of* assertions. In addition we also explain the intuition behind references on the left and right hand sides of rules.

LLMBRR models are supposed to be queried. For now we can think of a query as a request to obtain information about a concrete situation, such as what VAT rate to apply. In LLMBRR we use the **this** keyword to refer to the current situation. An answer to a query most likely depends on assumptions about the situation at hand, and we can think of rules as a way of specifying causal relationships between assumptions and conclusions. In LLMBRR assumptions as well as conclusions about the current situation (**this**) can be stated in three ways. The first way is to state that **this in** `ci` for some concept `ci`, which is to say that the current situation is an instance of or member in some concept. For example we can say **this in** `Goods` to state that the current situation is a supply of goods. The second (and third) way is to use aspect functions. An aspect function describes some aspect about the current situation (e.g., `sold(`**this**`)` **in** `Monarco` states that the current supply (situation) is a sale taking place in Monarco). The third way is to use aspect functions to build Presburger Arithmetic expressions such as `price(`**this**`)` = 75.

References

Another way of specifying assertions and conclusions in rules is to use references. References can be used on left (**lref**) as well as right (**rref**) hand sides of rules (Horn clauses) and is nothing but an easy way to duplicate the assertions respectively conclusions of the referenced rules (it is not allowed to reference exceptions). A priori it is possible to give semantics in the situation where references are cyclic, but since we do not think this is intended behaviour in legal documents we shall disallow cycles. Thus the set of references in a model will make up a forest of DAGs, where an arc between rules A and B means that A references B. The intended semantics is to unfold references stepwise such that a reference is unfolded in a given step iff the rules it references have already been unfolded.

More well-formedness requirements can be found in Appendix A.

5.1.3 Simplifying Assumptions

We have assumed that references can only be given to explicit sets of legal statement identifiers. Thus one cannot reference the collection of legal statements contained in a structure element (**legaldoc**, **legalsec**, **article** and **subarticle**) by referencing the id of that element. While this is partly a simplifying assumption it is also a design decision that we do not want to clutter the language with references that are tied to the hierarchical structure of the model. Instead relative references can (and should) be added as a feature of an IDE, which can use the structure (that we do model) to unfold relative references to absolute ones.

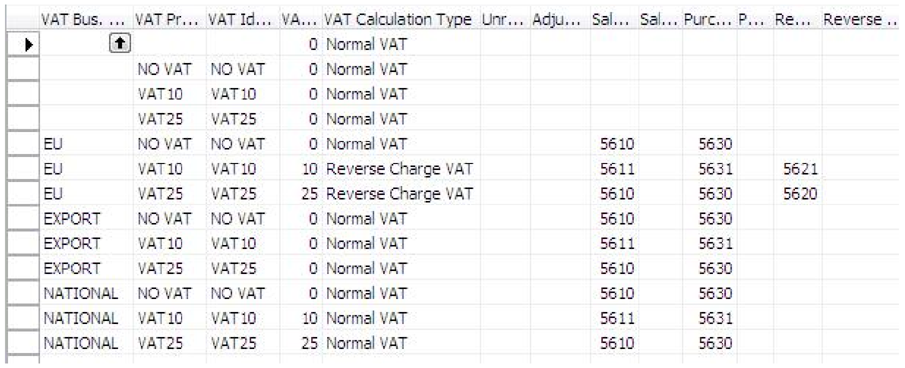
5.2 Examples

In this section we give two nontrivial examples of how LLMBRR can be used to model legal rules. Our first example demonstrates that LLMBRR can encode the user configurable part of the VAT setup in NAV and we argue that

LLMBRR scales better than the approach NAV uses today. Following that we present and evaluate an implementation of rules from a Danish guide to VAT on services [14].

5.2.1 Encoding VAT Rules From Microsoft Dynamics NAV

In this section we demonstrate that LLMBRR can encode the user configurable part of the VAT setup in Microsoft Dynamics NAV. As mentioned in Section 3.1 handling of VAT is set up in the Posting Setup matrix under Setup▷VAT Posting Group. Figure 5.3 shows the posting setup for CRONUS International Ltd. the built-in fictitious demo company distributed with NAV.



VAT Bus. ...	VAT Pr...	VAT Id...	VA...	VAT Calculation Type	Unr...	Adju...	Sal...	Sal...	Purc...	P...	Re...	Reverse ...
			0	Normal VAT								
	NO VAT	NO VAT	0	Normal VAT								
	VAT10	VAT10	0	Normal VAT								
	VAT25	VAT25	0	Normal VAT								
EU	NO VAT	NO VAT	0	Normal VAT			5610		5630			
EU	VAT10	VAT10	10	Reverse Charge VAT			5611		5631		5621	
EU	VAT25	VAT25	25	Reverse Charge VAT			5610		5630		5620	
EXPORT	NO VAT	NO VAT	0	Normal VAT			5610		5630			
EXPORT	VAT10	VAT10	0	Normal VAT			5611		5631			
EXPORT	VAT25	VAT25	0	Normal VAT			5610		5630			
NATIONAL	NO VAT	NO VAT	0	Normal VAT			5610		5630			
NATIONAL	VAT10	VAT10	10	Normal VAT			5611		5631			
NATIONAL	VAT25	VAT25	25	Normal VAT			5610		5630			

Fig. 5.3. Screen Dump of the Posting Setup of CRONUS International Ltd. in Microsoft Dynamics NAV 5.0.

As explained in Section 3.1 the posting setup is a partial map from VAT Business Posting Group and VAT Product Posting Group into an 11-tuple of values controlling the handling of VAT. In LLMBRR we encode each row in the figure as a rule. We model values as (disjoint) concepts and use aspect functions to model columns. Thus the LLMBRR code corresponding to `VAT Business Posting Group` having the value `EU` becomes `vatBusinessPG(this) in EU`.

In Figure 5.4 we have included the code corresponding to line 13 in Figure 5.3. The code consists of a single rule having as its condition that `vatBusinessPG(this) in NATIONAL and vatProductPG(this) in VAT25` and as its conclusion information similar to that in Figure 5.3 coded in the obvious way.

We have included a full LLMBRR model of the rules in row 7, 10 and 13 in Appendix B.1.1.

```
lstmt 9: vatBusinessPG(this) in NATIONAL and
                    vatProductPG(this) in VAT25
                    -> vatIdentifier(this) in VAT25 and
                       vatPercentage(this) = 25 and
                       adjustForPaymentDiscount(this) in False and
                       vatCalculationType(this) in Normal_VAT and
                       salesVatAccount(this) = 5610 and
                       purchaseVatAccount(this) = 5630
```

Fig. 5.4. LLMBRR code corresponding to line 13 of Figure 5.3.

Evaluation

We have seen that LLMBRR can model the configurable part of the VAT setup in NAV in a straight forward manner. The configurable part only determines the VAT scheme to use and the parameters (e.g., account numbers of sales and purchase accounts) it should be instantiated with. This is similar to determining the tax code in SAP with the exception that SAP takes a larger number of parameters into account.

On the other hand we have learned from staff at Microsoft Development Center Copenhagen that many country and region specific VAT customizations are hard-wired into the code. It would be interesting to see if LLMBRR can capture those issues as well and what the problems might be, but due to the reasons outlined in Chapter 3 we have not been able to do so.

The encoding in Appendix B.1.1 is rather large and clumsy compared with Figure 5.3. This is due the fact that LLMBRR is designed to encode relationships between values. For instance we see that the sales and purchase accounts are always `5610` respectively `5630` whenever the `VAT Product Posting Group` is `VAT25` and a lot of other dependencies can be discovered too. If we use such dependencies we arrive at the LLMBRR model in Appendix B.1.2. This model is still rather large compared to the posting setup. However, the posting setup must contain a row for each combination of `VAT Business Posting Group` and `VAT Product Posting Group`. For example if both columns had a fourth value (e.g., `NATO` and `VAT5`) then the posting setup would need to have sixteen rows instead of nine[3] even though the only differences might be that `VAT5` should be posted to different sales and purchase accounts and that `NATO` triggers another VAT calculation scheme. In LLMBRR these changes could be implemented using only two legal statements (lstmt).

In conclusion we expect LLMBRR models to scale much better than the posting setup.

5.2.2 Encoding Guidelines

Since legal guides can carry legal weight LLMBRR should be able to formalize their content. As a challenge we shall try to model parts of the Danish legal

[3] Rows 1, 2 and 3 are bogus and ought to be removed. They are there for demo purposes only.

guide *Moms af ydelser* (English: VAT on Services) [14]. This guide contains rules Danish companies engaged in the selling and buying of services across the border have to follow. The purpose of a model will be to determine when VAT is due on services and whether it is the responsibility of the buyer or the seller to calculate VAT.

Rules to Encode

The Guide is split into two parts. The first part contains the main rules, while the latter part lists a whole range of exceptions from the main rules. Since the guide [14] is in Danish we have translated central parts into English.

In order to answer questions about VAT on services one must know the *place of delivery* with respect to VAT, since it is that countrys VAT rules that apply. This country can be that of the seller; the buyer; or the country, where the service is used. In Denmark the main rule on how to determine the place of delivery with respect to VAT is:

> *Hovedreglen er, at Danmark betragtes som leveringssted for alle ydelser, som virksomheder etableret her i landet sælger. Dvs. at virksomheder etableret i Danmark skal beregne dansk moms, når de sælger momspligtige ydelser, uanset hvem køberen er. Det har ingen betydning, om ydelsen rent fysisk leveres her i landet eller i udlandet.*
>
> [14, p. 8]

And translated into English:

> *The main rule is that Denmark is the place of delivery with respect to VAT for all services rendered by companies established in Denmark. Thus companies established in Denmark have to calculate Danish VAT when they sell services subject to VAT no matter who the buyer is. Whether the service is rendered physically in Denmark or abroad is without significance.*
>
> Translated from [14, p. 8]

In addition the rule also establishes that the seller must calculate Danish VAT. The following rule establishes the standard VAT percentage on services:

> *I Danmark er alle ydelser som udgangspunkt momspligtige, og for de fleste momspligtige ydelser gælder den almindelige momssats på 25 pct.*
>
> [14, p. 10]

And translated into English:

> *As a rule all services are subject to VAT in Denmark. For most services sujbect to VAT the standard rate (25 pct) applies.*

Translated from [14, p. 10]

A LLMBRR model of the main rules is included in Appendix B.2 (lstmt 1 - lstmt 3).

Exceptions

As mentioned the second part of the guide contains a list of exceptions to the main rules. Here we only focus on *electronic services*, since the other exceptions are similar. The rules governing selling and buying of electronic services are presented in schematic form in Figure 5.5.

A careful analysis of the rules reveals that it is not all rows that state proper exceptions to the main rules in the sense that their conclusions disagree with the main rules. In fact the only proper exceptions in the first tabular are rows 2 and 4 (corresponding to lstmt 7 and 10 in our model in Appendix B.2). However we can safely declare rows 1 and 3 to be exceptions to the part of the main rule having to do with place of delivery (lstmt 1) because of the way semantics of exceptions is intended to work (according to Section 2.3.1). In the second tabular the proper exceptions correspond to rows 2 and 4, but this time they are exceptions to the part of the main rule deciding who has to calculate VAT (lstmt 3). Of the other rows in the second tabular the first one (pendant to row 1 one of the first tabular) could be stated as an exception to the main rule governing place of delivery (lstmt 1) but the rows 3 and 5 must not be stated as exceptions to any part of the main rule (because they are not).

With respect to implementation we could have chosen to leave improper exceptions out and not to include the non-exceptions in rows 3 and 5 of the second tabular. We have chosen to include them here for consistency reasons.

Evaluation

It has been relatively straight forward to encode the above rules about VAT on services in LLMBRR, aside from the fact that it is impossible to express *Not Denmark* on the right hand side of rules. A priori this problem can be fixed in at least two ways. One way is to allow for negation on right hands sides of *Rule* expressions just as negation can be used on left hand sides, while another way is to include a construction for concept (set) difference. In the latter case *Not Denmark* could have been encoded as `AnotherEuState = EU minus Denmark` and then we could have used `AnotherEuState` on the right hand side.

Another issue, which became evident during the development of the code in Appendix B.2 is the importance of tool support such as an IDE with built in reasoning capabilities. The reason is that it is not easy at all to state the right (implicit) relationships between concepts, especially in the case of disjointness. Finally it is also hard to find out whether a new rule to be added is in conflict

Selling of Electronic Services		
Who is the buyer?	**Place of delivery?**	**Who calculates VAT?**
Private or taxable person in Denmark	Denmark	The seller must calculate Danish VAT.
Taxable person in another EU-state	Not Denmark	The seller should not calculate VAT, but note that the *reverse charge* VAT scheme applies.
Private person in another EU-state	Denmark	The seller must calculate Danish VAT.
Private or taxable person outside EU	Not Denmark	The seller should not calculate VAT.

Buying of Electronic Services			
Who is the seller?	**Who is the buyer?**	**Place of delivery?**	**Who calculates VAT?**
Company in Denmark	Private or taxable person in Denmark	Denmark	The seller must calculate Danish VAT.
Company in another EU-state	Taxable person in Denmark	Denmark	The buyer must calculate Danish VAT.
	Private person in Denmark	Not Denmark	Neither the buyer nor the seller should calculate Danish VAT.
Company outside EU	Taxable person in Denmark	Denmark	The buyer must calculate Danish VAT.
	Private person in Denmark	Denmark	The seller must calculate Danish VAT.

Fig. 5.5. Danish rules governing the selling and buying of electronic services. The rules are translated from [14, p. 44-45].

with (and thus should be an exception to) existing rules, so tool support is desirable for this task as well.

5.3 Legal Reasoning in the Context of LLMBRR

In this section we illustrate by way of example how reasoning about LLMBRR models should be performed from a domain experts point of view. To this end we shall consider the LLMBRR model in Figure 5.6 of the strawberry example from Section 2.3.

```
model
  legaldoc "Strawberry Model"
    header
      enacted: "01:01:2008"
      importance-level: 1
    body
      legalsec 1 "Implicit Definitions"
        article 1
          lstmt 1: implicit OrganicFV sub Groceries
          lstmt 2: implicit all_disjoint { StdVATRate, ZeroVATRate }
      end
      legalsec 2 "VAT Rates"
        article 2
          lstmt 3: this in StdVATRate -> vatRate(this) = 25
          lstmt 4: this in ZeroVATRate -> vatRate(this) = 0
      end
      legalsec 3 "Rules"
        article 3
          lstmt 5: this in Groceries -> this in StdVATRate
          lstmt 6: this in OrganicFV -> this in ZeroVATRate
      end
end
```

Fig. 5.6. LLMBRR model corresponding to the strawberries example in Section 2.3.

The model consists of two implicit definitions (lstmt 1 and 2) and four rules (lstmt 3-6). Suppose that we buy a punnet of organic strawberries and that we want to query the model in order to determine the VAT rate to apply. First of all we take $A = \{$**this in** `OrganicFV`$\}$ to be our set of assumptions and we *tell* (how is not important) the system that we want an answer to `vatRate`(**this**) as a response.

Following the pseudo algorithms in Figure 4.1 the first thing to do is to determine the set of *applicable* rules: that is, rules whose left hand side is satisfied. Using the definition in lstmt 1 we conclude that **this in** `Groceries`. Thus the set of applicable rules is $\mathcal{R}_1 = \{r_5, r_6\}$, where r_5 and r_6 refer to the rules in lstmt 5 respectively 6.

The next step is to disambiguate $\mathcal{R}_1$. Clearly we have a conflict, because `StdVATRate` and `ZeroVATRate` are required to be disjoint in lstmt 2 and then it cannot be that **this in** `StdVATRate` and **this in** `ZeroVATRate` at the same time. According to the pseudo algorithm for disambiguation of rules (Figure 2.5) we should apply Lex Superior first of all. However both of the rules are equally important, because they belong to the same document (and thus they have the same importance-level, which is 1 by the way). Pruning with respect to main rules and exceptions does not help either so Lex Specialis and Lex Posterior must be used. Lex Posterior assigns precedence based on the enactment dates of the legal rules in question, but as before the rules belong to the same document and thus that date is the same too. Returning to Lex Specialis however, we can use the model to conclude that `OrganicFV` is contained in `Groceries` (lstmt 1) and thus that rule r_6 is more specific than r_5. Therefore the disambiguation pseudo algorithm only returns rule r_6 together with an indication that an instructive reasoning principle has been used.

The next step of the pseudo reasoning algorithm is to add the conclusions of rule r_6 to A in order to obtain $A' = \{\textbf{this in } \texttt{OrganicFV}, \textbf{this in } \texttt{ZeroVATRate}\}$ before it calls itself recursively.

Assuming that we use ReasonNoCyc the next iteration will only be able to chose between rules r_3 and r_4 since r_5 and r_6 have already been used. The only rule that applies is r_4 and thus we get
$A'' = \{\textbf{this in } \texttt{OrganicFV}, \textbf{this in } \texttt{ZeroVATRate}, \texttt{vatRate}(\textbf{this}) = 0\}$, which is the answer returned from the reasoning algorithm when it terminates (together with an indication that reasoning was instructive). As a final step the system could restrict A'' to the set of answers the user requested. Thus an answer could have the form $C = \{\texttt{vatRate}(\textbf{this}) = 0\}$.

5.3.1 Comments

The example provided here is extremely simple, but it illustrates the step-by-step fashion in which domain experts think about and perform reasoning. Even though many details remain to be worked out it seems that LLMBRR could be given a *query semantics* (for lack of a better term) - simply describing how a model "responds" to a query. While a query semantics would enable us to implement a reasoner for the language it is not the best choice, when it comes to automated analysis. Instead we would like a semantics based on a well understood logic. For instance it would be nice to be able to describe the set of *knowledge* obtainable from a model M given a set of assumptions A by a fixed point such as $\cup_{i=0}^{\omega}\mathcal{F}(A)$, where $F : \mathrm{dom}\,(A) \to \mathrm{dom}\,(A)$ is a function "applying" the model M to the set of assumptions A much like we did above.

6

Summary & Future Work

This is the final chapter. It is time to summarize the work done so far, and to outline directions for future work.

6.1 Summary

Our investigations of VAT in an ERP context began as a response to real world problems put forward by Microsoft in relation to Microsoft Dynamics NAV (an ERP system for SMEs). By looking into NAV (and SAP) we found that VAT support rendered by ERP systems is often oversimplified and furthermore that it is very difficult to get an overview of what rules an ERP system implements. The problems with handling of VAT in ERP systems taken together with the immense trust we put in them (consumers, accountants, business owners and auditors alike) motivated us to investigate to what extend relevant legal rules can be modeled using an approach based on logics. From the beginning it has been our belief that modeling should (ideally) be undertaken by domain experts - one of the benefits being a reduced need to create informal program/feature specifications (see the introduction for details).

In the works preceding this report we have experimented with various logic based approaches to legal rule modeling, which led to the conclusion that no domains specific language (and reasoning service) exists for the legal domain[1].

This report began with an analysis of the legal domain, which amongst others explained the principles behind legal reasoning. Furthermore we have described how VAT is supported in NAV and SAP. We concluded the first part of this report with a list of requirements on the design of a DSL for legal rule modeling. In part two we have discussed the design of such a language from a practical point of view: We have illustrated how query semantics can

[1] As described in the introduction various logics do exist, but their aim is somewhat different from ours.

be given by way of example, and we have seen that our “language”, LLMBRR, can encode the user configurable part of the VAT support in NAV.

6.2 Looking Forward

The greatest weakness in this report is that we have not had the time to finalize the second part. Looking forward we want to research logics for knowledge representation including an analysis comparing their adequacy with respect to legal modeling. That analysis will require a metric taking important parameters such as support for: Presburger Arithmetic, decidable reasoning, isomorphic modeling, and separation of main rules and their exceptions into account to name a few.

Once we have investigated the logics deemed interesting we should reiterate our language design based on results obtained from the analysis, and we should investigate how a query semantics relates to (possibly several kinds of) logic based semantics. Finally we should design a software component with a well-defined query interface (e.g., as a web service) allowing us to build and expose model driven query engines to ERP and other systems.

A

Well-formed LLMBRR

Remark A.1. It would be premature to include this section in the main matter, but nevertheless it conveys some views that might be of interest. For that reason we have chosen to include it in the appendix, even though the content is very much subject to change in the future.

Having discussed various aspects of LLMBRR informally and having seen an example of how LLMBRR can be used to model a legal source we are almost ready to give formal semantics to the language. As mentioned we only give semantics to well-formed LLMBRR models. In this section we state our well-formedness requirements and make the notion precise.

First of all we shall require that `di`'s are unique on a model wide basis, that `ai`'s and `li`'s are unique within their surounding **legaldoc** and that `lsi`'s and `subi`'s are unique within their enclosing construction. In addition we shall require the following:

1. References must not be dangling. That is, **lref**, **rref** and **exception_to** must not reference legal statements that do not exist.
2. Only legal statements of the type *Rule* may be referenced via **lref** and **rref**. In addition **exception_to** statements may also reference other exceptions.
3. References must not be cyclic.
4. Concepts and Aspect Functions mentioned in the *Aliases* part must be (implicitly) defined.
5. Aspect Functions must be used in a type consistent way. That is, aspect functions used in Presburger Arithmetic expressions cannot be used in *element of* assertions and vice versa.

All of these requirements can be made precise using well-formedness relations. For instance we could use one relation to ensure the uniqueness requirements and another to ensure requirements 1 - 5. Well-formedness relations will necessarily contain a lot of rules because of the size of the syntax. Therefore we have decided only to state the latter (and more interesting) of the two.

Definition A.2. *The well-formedness relation $\vdash_{wf} \cdot : \cdot$ for core LLMBRR models satisfying that* **`id`***'s are unique on a model wide basis, that* **`ai`***'s and* **`li`***'s are unique within their surounding* ***legaldoc*** *and that* **`lsi`***'s and* **`subi`***'s are unique within their enclosing construction is defined to be the least relation satisfying the inference rules in Figures A.2, A.3 and A.4. We say that a LLMBRR model m satisfying the uniqueness requirements is* well-formed *with signature $(\varDelta, \varLambda_c, \varLambda_p)$, where $\varDelta \subseteq Concepts_{id}$ and $\varLambda_c, \varLambda_p \subseteq Afun_{id}$ iff $\vdash_{wf} m : (\varDelta, \varLambda_c, \varLambda_p)$.*

Remark A.3. Most of the rules in Figures A.2, A.3 and A.4 are used to traverse the structure of LLMBRR models in order to build the family of sets $Sig_i = (\varDelta_i, \varLambda_{c_i}, \varLambda_{p_i}, \varOmega_{R_i}, \varOmega_{E_i}, \varUpsilon_{L_i}, \varUpsilon_{R_i}, \varPsi_i)$, (part of) which is used in the side conditions of the rules MODEL-1, MODEL-2, CON and AFUN in order to enforce the well-formedness requirements presented above. The intention is that $\varDelta$'s are sets of used concept identifiers, $\varLambda_p$'s and $\varLambda_c$'s are sets of aspect function identifiers used in Presburger Arithmetic expressions and *element of* assertions respectively. $\varOmega_R$'s and $\varOmega_E$'s are sets of used legal statement identifiers for rules and exceptions respectively. Finally $\varUpsilon_L$'s, $\varUpsilon_R$'s and $\varPsi$'s are sets of pairs of legal statement identifiers where (l, r) is a member iff the legal statement identified by l references the legal statement identified by r via **lref**, **rref** and **exception_to** respectively.

The tabular in Figure A.1 shows for each well-formedness requirement the combination of side conditions on rules that realize it. We note that some of the side conditions could be collapsed into one line expressions, but this has been avoided due to formatting issues (line width).

Side Condition	**Req. 1**	**Req. 2**	**Req. 3**	**Req. 4**	**Req. 5**
MODEL :					
$\cup_i (\varOmega_{R_i} \cup \varOmega_{E_i}) \supseteq \pi_2 \cup_i \varPsi_i$	×	×			
$\cup_i \varOmega_{R_i} \supseteq \pi_2 \cup_i (\varUpsilon_{L_i} \cup \varUpsilon_{R_i})$	×	×			
$(\cup_i \varUpsilon_{L_i})^+ \cap \{(l,l) \mid l \in Lstmt_{id}\} = \emptyset$			×		
$(\cup_i \varUpsilon_{R_i})^+ \cap \{(l,l) \mid l \in Lstmt_{id}\} = \emptyset$			×		
$(\cup_i \varPsi_i)^+ \cap \{(l,l) \mid l \in Lstmt_{id}\} = \emptyset$			×		
$(\cup_i \varLambda_{c_i}) \cap (\cup_i \varLambda_{p_i}) = \emptyset$					×
CON :					
$\{\texttt{di.ci}, \texttt{di'.ci'}\} \subseteq \varDelta$				×	
AFUN :					
$\{\texttt{di.afid}, \texttt{di'.afid'}\} \subseteq \varLambda_c \vee$ $\{\texttt{di.afid}, \texttt{di'.afid'}\} \subseteq \varLambda_p$				×	×

Fig. A.1. The tabular shows how combinations of side conditions of inference rules work to satisfy the well-formedness requirements.

Lemma A.4. *Let m be any LLMBRR model. If $\vdash_{wf} m : (\varDelta, \varLambda_c, \varLambda_p)$ and $\vdash_{wf} m : (\varDelta', \varLambda'_c, \varLambda'_p)$ then $\varDelta = \varDelta'$, $\varLambda_c = \varLambda'_c$ and $\varLambda_p = \varLambda'_p$. That is, the well-*

$$\textsf{MODEL-1}\ \frac{\forall 1 \le i \le n.\ \vdash_{wf'} LD_i : (\Delta_i, \Lambda_{c_i}, \Lambda_{p_i}, \Omega_{R_i}, \Omega_{E_i}, \Upsilon_{L_i}, \Upsilon_{R_i}, \Psi_i)}{\vdash_{wf} \textbf{model}\ LD_1 \cdots LD_n\ \textbf{end} : (\cup_i \Delta_i, \cup_i \Lambda_{c_i}, \cup_i \Lambda_{p_i})}\ \begin{array}{c} \cup_i (\Omega_{R_i} \cup \Omega_{E_i}) \supseteq \pi_2 \cup_i \Psi_i \\ \cup_i \Omega_{R_i} \supseteq \pi_2 \cup_i (\Upsilon_{L_i} \cup \Upsilon_{R_i}) \\ (\cup_i \Upsilon_{L_i})^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Upsilon_{R_i})^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Psi_i)^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Lambda_{c_i}) \cap (\cup_i \Lambda_{p_i}) = \emptyset \end{array}$$

$$\textsf{MODEL-2}\ \frac{\begin{array}{c}\cup_i \Delta_i, \cup_i \Lambda_{c_i}, \cup_i \Lambda_{p_i} \vdash_{alwf} \mathit{Aliases} \\ \forall 1 \le i \le n.\ \vdash_{wf'} LD_i : (\Delta_i, \Lambda_{c_i}, \Lambda_{p_i}, \Omega_{R_i}, \Omega_{E_i}, \Upsilon_{L_i}, \Upsilon_{R_i}, \Psi_i)\end{array}}{\vdash_{wf} \textbf{model}\ LD_1 \cdots LD_n\ \mathit{Aliases}\ \textbf{end} : (\cup_i \Delta_i, \cup_i \Lambda_{c_i}, \cup_i \Lambda_{p_i})}\ \begin{array}{c} \cup_i (\Omega_{R_i} \cup \Omega_{E_i}) \supseteq \pi_2 \cup_i \Psi_i \\ \cup_i \Omega_{R_i} \supseteq \pi_2 \cup_i (\Upsilon_{L_i} \cup \Upsilon_{R_i}) \\ (\cup_i \Upsilon_{L_i})^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Upsilon_{R_i})^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Psi_i)^+ \cap \{(l,l) | l \in \mathit{Lstmt}_{id}\} = \emptyset \\ (\cup_i \Lambda_{c_i}) \cap (\cup_i \Lambda_{p_i}) = \emptyset \end{array}$$

$$\textsf{ALIAS}\ \frac{\Delta \vdash_{con} \mathit{ConceptAl} \qquad \Lambda_c, \Lambda_p \vdash_{af} \mathit{AFAl}}{\Delta, \Lambda_c, \Lambda_p \vdash_{alwf} \textbf{aliases concepts:}\ \mathit{ConceptAl}\ \textbf{aspectfunctions:}\ \mathit{AFAl}}$$

$$\textsf{CON}\ \frac{\Delta \vdash_{con} \mathit{ConceptAl}}{\Delta \vdash_{con} \texttt{di.ci}\ \textbf{sub}\ \texttt{di'.ci'},\ \mathit{ConceptAl}}\ \{\texttt{di.ci}, \texttt{di'.ci'}\} \subseteq \Delta \qquad \textsf{CON-}\emptyset\ \frac{}{\Delta \vdash_{con} \cdot}$$

$$\textsf{AFUN}\ \frac{\Lambda_c, \Lambda_p \vdash_{af} \mathit{AFAl}}{\Lambda_c, \Lambda_p \vdash_{af} \texttt{di.afid}\ \textbf{eq}\ \texttt{di'.afid'},\ \mathit{AFAl}}\ \begin{array}{c}\{\texttt{di.afid}, \texttt{di'.afid'}\} \subseteq \Lambda_c \vee \\ \{\texttt{di.afid}, \texttt{di'.afid'}\} \subseteq \Lambda_p\end{array} \qquad \textsf{AFUN-}\emptyset\ \frac{}{\Lambda_c, \Lambda_p \vdash_{af} \cdot}$$

Fig. A.2. Top-level and Alias rules of the Well-Formedness Relation for Core LLM-BRR. The $\vdash_{wf'} \cdot : \cdot$ relation is defined in Figure A.3.

formedness relation is functional. Moreover if p *is a derivation of* $\vdash_{wf} m : (\Delta, \Lambda_c, \Lambda_p)$ *and* p' *is a derivation of* $\vdash_{wf} m : (\Delta', \Lambda'_c, \Lambda'_p)$ *then* $p = p'$*.*

The proof, which is a simple (and tedious) induction on the structure of m, follows from the observation that at each step in a derivation there is exactly one inference rule, which can be applied

$$\textsf{LEGALDOC}\ \frac{\texttt{di} \vdash_{wfb} \mathit{Body} : \mathit{Sig}}{\vdash_{wf'} \textbf{legaldoc}\ \texttt{di}\ \mathit{Header}\ \ \mathit{Body} : \mathit{Sig}}$$

$$\textsf{LD-BODY}\ \frac{\forall 1 \leq i \leq n . \texttt{di} \vdash_{wfls} \mathit{Legalsec}_i : \mathit{Sig}_i}{\texttt{di} \vdash_{wfb} \textbf{body}\ \mathit{Legalsec}_1 \cdots \mathit{Legalsec}_n : \cup_i \mathit{Sig}_i}$$

$$\textsf{LEGALSEC}\ \frac{\texttt{di} \vdash_{wflsb} \mathit{LSBody} : \mathit{Sig}}{\texttt{di} \vdash_{wfls} \textbf{legalsec}\ \texttt{lsi}\ \mathit{Optname}\ \ \mathit{LSBody}\ \textbf{end} : \mathit{Sig}}$$

$$\textsf{LS-BODY-1}\ \frac{\forall 1 \leq i \leq n . \texttt{di} \vdash_{wfls} \mathit{Legalsec}_i : \mathit{Sig}_i}{\texttt{di} \vdash_{wflsb} \mathit{Legalsec}_1 \cdots \mathit{Legalsec}_n : \cup_i \mathit{Sig}_i}$$

$$\textsf{LS-BODY-2}\ \frac{\forall 1 \leq i \leq n . \texttt{di} \vdash_{wfa} \mathit{Article}_i : \mathit{Sig}_i}{\texttt{di} \vdash_{wflsb} \mathit{Article}_1 \cdots \mathit{Article}_n : \cup_i \mathit{Sig}_i}$$

$$\textsf{ARTICLE}\ \frac{\texttt{di} \vdash_{wfsab} \mathit{SABody} : \mathit{Sig}}{\texttt{di} \vdash_{wfa} \textbf{article}\ \texttt{ai}\ \mathit{SABody} : \mathit{Sig}}$$

$$\textsf{SA-BODY-1}\ \frac{\forall 1 \leq i \leq n . \texttt{di} \vdash_{wfsa} \mathit{Subarticle}_i : \mathit{Sig}_i}{\texttt{di} \vdash_{wfsab} \mathit{Subarticle}_1 \cdots \mathit{Subarticle}_n : \cup_i \mathit{Sig}_i}$$

$$\textsf{SA-BODY-2}\ \frac{\forall 1 \leq i \leq n . \texttt{di} \vdash_{wfl} \mathit{Lstmt}_i : \mathit{Sig}_i}{\texttt{di} \vdash_{wfsab} \mathit{Lstmt}_1 \cdots \mathit{Lstmt}_n : \cup_i \mathit{Sig}_i}$$

$$\textsf{SUBARTICLE}\ \frac{\texttt{di} \vdash_{wfsab} \mathit{SABody} : \mathit{Sig}}{\texttt{di} \vdash_{wfsa} \textbf{subarticle}\ \texttt{subi}\ \mathit{SABody}\ \textbf{end} : \mathit{Sig}}$$

Fig. A.3. Structural rules of the Well-Formedness Relation for Core LLMBRR. The $\cdot \vdash_{wfl} \cdot : \cdot$ relation is defined in Figure A.4.

$$\textsf{IDEF}\quad\frac{\texttt{di} \vdash_{wfdef} \mathit{Def} : \Delta}{\texttt{di} \vdash_{wfl} \textbf{lstmt } \texttt{li} \textbf{ : implicit } \mathit{Def} : (\Delta,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}$$

$$\textsf{DEF}\quad\frac{\texttt{di} \vdash_{wfdef} \mathit{Def} : \Delta}{\texttt{di} \vdash_{wfl} \textbf{lstmt } \texttt{li} \textbf{ : } \mathit{Def} : (\Delta,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}
\qquad
\textsf{DEF-AD}\quad\frac{\texttt{di} \vdash_{wfcis} \mathit{Cis} : \Delta}{\texttt{di} \vdash_{wfdef} \textbf{all\_disjoint } \{\mathit{Cis}\} : \Delta}$$

$$\textsf{DEF-S}\quad\frac{\texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta_1 \qquad \texttt{di} \vdash_{wfcis} \mathit{Cis} : \Delta_2}{\texttt{di} \vdash_{wfdef} \mathit{Ci} \textbf{ sub } \mathbf{U}\{\mathit{Cis}\} : \Delta_1 \cup \Delta_2}
\qquad
\textsf{CIS-1}\quad\frac{}{\texttt{di} \vdash_{wfcis} \texttt{di'.ci'} : \{\texttt{di'.ci'}\}}$$

$$\textsf{DEFDEF}\quad\frac{\texttt{di} \vdash_{wfdef} \mathit{Def_1} : \Delta_1 \qquad \texttt{di} \vdash_{wfdef} \mathit{Def_2} : \Delta_2}{\texttt{di} \vdash_{wfdef} \mathit{Def_1}\textbf{;}\ \mathit{Def_2} : \Delta_1 \cup \Delta_2}
\qquad
\textsf{CIS-2}\quad\frac{}{\texttt{di} \vdash_{wfcis} \texttt{ci} : \{\texttt{di.ci}\}}$$

$$\textsf{CIS-3}\quad\frac{\texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta_1 \qquad \texttt{di} \vdash_{wfcis} \mathit{Cis} : \Delta_2}{\texttt{di} \vdash_{wfcis} \mathit{Ci},\ \mathit{Cis} : \Delta_1 \cup \Delta_2}$$

$$\textsf{EXN}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflis} \mathit{Lis} : \Psi \qquad \texttt{di},\texttt{li} \vdash_{wflhs} \mathit{LHS} : \mathit{Sig_1} \qquad \texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{RHS} : \mathit{Sig_2}}{\texttt{di} \vdash_{wfl} \textbf{lstmt } \texttt{li} \textbf{ : exception\_to } \{\mathit{Lis}\}\ \mathit{LHS} \ {-}{>}\ \mathit{RHS} : \mathit{Sig_1} \cup \mathit{Sig_2} \cup \{\emptyset,\emptyset,\emptyset,\emptyset,\{\texttt{di.li}\},\emptyset,\emptyset,\Psi\}}$$

$$\textsf{RULE}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflhs} \mathit{LHS} : \mathit{Sig_1} \qquad \texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{RHS} : \mathit{Sig_2}}{\texttt{di} \vdash_{wfl} \textbf{lstmt } \texttt{li} \textbf{ : } \mathit{LHS} \ {-}{>}\ \mathit{RHS} : \mathit{Sig_1} \cup \mathit{Sig_2} \cup \{\emptyset,\emptyset,\emptyset,\{\texttt{di.li}\},\emptyset,\emptyset,\emptyset,\emptyset\}}$$

$$\textsf{LIS-1}\quad\frac{}{\texttt{di},\texttt{li} \vdash_{wflis} \texttt{di'.li'} : \{(\texttt{di.li},\texttt{di'.li'})\}}
\qquad
\textsf{LIS-2}\quad\frac{}{\texttt{di},\texttt{li} \vdash_{wflis} \texttt{li'} : \{(\texttt{di.li},\texttt{di.li'})\}}$$

$$\textsf{LIS-3}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflis} \mathit{Li} : X_1 \qquad \texttt{di},\texttt{li} \vdash_{wflis} \mathit{Lis} : X_2}{\texttt{di},\texttt{li} \vdash_{wflis} \mathit{Li},\ \mathit{Lis} : X_1 \cup X_2}
\qquad
\textsf{LEXPR}\quad\frac{\texttt{di} \vdash_{wflexpr} \mathit{Lexpr} : \mathit{Sig}}{\texttt{di},\texttt{li} \vdash_{wflhs} \mathit{Lexpr} : \mathit{Sig}}$$

$$\textsf{LREF}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflis} \mathit{Lis} : \Upsilon_L}{\texttt{di},\texttt{li} \vdash_{wflhs} \textbf{lref } \{\mathit{Lis}\} : (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\Upsilon_L,\emptyset,\emptyset)}
\qquad
\textsf{RREF}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflis} \mathit{Lis} : \Upsilon_R}{\texttt{di},\texttt{li} \vdash_{wfrhs} \textbf{rref } \{\mathit{Lis}\} : (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\Upsilon_R,\emptyset)}$$

$$\textsf{LAND}\quad\frac{\texttt{di},\texttt{li} \vdash_{wflhs} \mathit{LHS_1} : \mathit{Sig_1} \qquad \texttt{di},\texttt{li} \vdash_{wflhs} \mathit{LHS_2} : \mathit{Sig_2}}{\texttt{di},\texttt{li} \vdash_{wflhs} \mathit{LHS_1} \textbf{ and } \mathit{LHS_2} : \mathit{Sig_1} \cup \mathit{Sig_2}}
\qquad
\textsf{REXPR}\quad\frac{\texttt{di} \vdash_{wfrexpr} \mathit{Rexpr} : \mathit{Sig}}{\texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{Rexpr} : \mathit{Sig}}$$

$$\textsf{RAND}\quad\frac{\texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{RHS_1} : \mathit{Sig_1} \qquad \texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{RHS_2} : \mathit{Sig_2}}{\texttt{di},\texttt{li} \vdash_{wfrhs} \mathit{RHS_1} \textbf{ and } \mathit{RHS_2} : \mathit{Sig_1} \cup \mathit{Sig_2}}
\qquad
\textsf{LNOT}\quad\frac{\texttt{di} \vdash_{wflexpr} \mathit{Lexpr} : \mathit{Sig}}{\texttt{di} \vdash_{wflexpr} \textbf{not } \mathit{Lexpr} : \mathit{Sig}}$$

$$\textsf{LPRES}\quad\frac{\texttt{di} \vdash_{wfpresb} \mathit{Presb_1} : \mathit{Sig_1} \qquad \texttt{di} \vdash_{wfpresb} \mathit{Presb_2} : \mathit{Sig_2}}{\texttt{di} \vdash_{wflexpr} \mathit{Presb_1} \ < \ \mathit{Presb_2} : \mathit{Sig_1} \cup \mathit{Sig_2}}$$

$$\textsf{L-IN-1}\quad\frac{\texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta}{\texttt{di} \vdash_{wflexpr} \textbf{this in } \mathit{Ci} : (\Delta,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}
\qquad
\textsf{L-IN-2}\quad\frac{\texttt{di} \vdash_{wfafid} \mathit{Afid} : \Lambda_c \qquad \texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta}{\texttt{di} \vdash_{wflexpr} \mathit{Afid}\textbf{(this) in } \mathit{Ci} : (\Delta,\Lambda_c,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}$$

$$\textsf{AFID-1}\quad\frac{}{\texttt{di} \vdash_{wfafid} \texttt{di'.afid'} : \{\texttt{di'.afid'}\}}
\qquad
\textsf{AFID-2}\quad\frac{}{\texttt{di} \vdash_{wfafid} \texttt{afid} : \{\texttt{di.afid}\}}$$

$$\textsf{RPRES}\quad\frac{\texttt{di} \vdash_{wfpresb} \mathit{Presb_1} : \mathit{Sig_1} \qquad \texttt{di} \vdash_{wfpresb} \mathit{Presb_2} : \mathit{Sig_2}}{\texttt{di} \vdash_{wfrexpr} \mathit{Presb_1} \ = \ \mathit{Presb_2} : \mathit{Sig_1} \cup \mathit{Sig_2}}$$

$$\textsf{R-IN-1}\quad\frac{\texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta}{\texttt{di} \vdash_{wfrexpr} \textbf{this in } \mathit{Ci} : (\Delta,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}
\qquad
\textsf{R-IN-2}\quad\frac{\texttt{di} \vdash_{wfafid} \mathit{Afid} : \Lambda_c \qquad \texttt{di} \vdash_{wfcis} \mathit{Ci} : \Delta}{\texttt{di} \vdash_{wfrexpr} \mathit{Afid}\textbf{(this) in } \mathit{Ci} : (\Delta,\Lambda_c,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}$$

$$\textsf{PRESB-1}\quad\frac{\texttt{di} \vdash_{wfafid} \mathit{Afid} : \Lambda_p}{\texttt{di} \vdash_{wfpresb} \mathit{Afid}\textbf{(this)} : (\emptyset,\emptyset,\Lambda_p,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}
\qquad
\textsf{PRESB-2}\quad\frac{}{\texttt{di} \vdash_{wfpresb} \overline{\texttt{c}} : (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset)}$$

$$\textsf{PRESB-3}\quad\frac{\texttt{di} \vdash_{wfpresb} \mathit{Presb_1} : \mathit{Sig_1} \qquad \texttt{di} \vdash_{wfpresb} \mathit{Presb_2} : \mathit{Sig_2}}{\texttt{di} \vdash_{wfpresb} \mathit{Presb_1} \ + \ \mathit{Presb_2} : \mathit{Sig_1} \cup \mathit{Sig_2}}$$

Fig. A.4. Signature Building Rules of the Well-Formedness Relation for Core LLM-BRR.

B

Example Code

B.1 LLMBRR Encoding of VAT Rules From Microsoft Dynamics NAV

Below we present two LLMBRR encodings of the conditions in row 7, 10 and 13 of Figure 5.3. The first is a straight forward translation, while the latter exploits dependencies between values.

B.1.1 Straight Forward Encoding

```
model
  legaldoc "CRONUS International Ltd – Microsoft Dynamics NAV – VAT Model"
    header
      enacted: "01:01:2008"
      importance–level: 3
    body
      legalsec 1 "VAT Posting Setup"
        legalsec 1 "Implicit assumptions"
          article 1
            lstmt 1: implicit all_disjoint {EU, EXPORT, NATIONAL}

          article 2
            lstmt 2: implicit all_disjoint {NO_VAT, VAT10, VAT25}

          article 3
            lstmt 3: implicit all_disjoint {Percentage, First, Last,
                                            FirstFullyPaid, LastFullyPaid}

          article 4
            lstmt 4: implicit all_disjoint {True, False}

          article 5
            lstmt 5: implicit all_disjoint {Normal_VAT, Reverse_Charge_VAT,
                                            Full_VAT, Sales_Tax}
        end
        legalsec 2 "Rules"
          article 7
            lstmt 7: vatBusinessPG(this) in EU and
                     vatProductPG(this) in VAT25
                     –> vatIdentifier(this) in VAT25 and
                        vatPercentage(this) = 25 and
```

```
                            adjustForPaymentDiscount(this) in False and
                            vatCalculationType(this) in Reverse_Charge_VAT and
                            salesVatAccount(this) = 5610 and
                            purchaseVatAccount(this) = 5630 and
                            reverseChargeVatAccount(this) = 5620

             lstmt 8: vatBusinessPG(this) in EXPORT and
                      vatProductPG(this) in VAT25
                      -> vatIdentifier(this) in VAT25 and
                         vatPercentage(this) = 0 and
                         adjustForPaymentDiscount(this) in False and
                         vatCalculationType(this) in Normal_VAT and
                         salesVatAccount(this) = 5610 and
                         purchaseVatAccount(this) = 5630

             lstmt 9: vatBusinessPG(this) in NATIONAL and
                      vatProductPG(this) in VAT25
                      -> vatIdentifier(this) in VAT25 and
                         vatPercentage(this) = 25 and
                         adjustForPaymentDiscount(this) in False and
                         vatCalculationType(this) in Normal_VAT and
                         salesVatAccount(this) = 5610 and
                         purchaseVatAccount(this) = 5630
        end
      end
end
```

B.1.2 Encoding Using Dependencies Between Values

```
model
  legaldoc "CRONUS International Ltd - Microsoft Dynamics NAV - VAT Model 2"
    header
      enacted: "01:01:2008"
      importance-level: 3
    body
      legalsec 1 "VAT Posting Setup"
        legalsec 1 "Implicit assumptions"
          article 1
            lstmt 1: implicit all_disjoint {EU, EXPORT, NATIONAL}

          article 2
            lstmt 2: implicit all_disjoint {NO_VAT, VAT10, VAT25}

          article 3
            lstmt 3: implicit all_disjoint {Percentage, First, Last,
                                            FirstFullyPaid, LastFullyPaid}

          article 4
            lstmt 4: implicit all_disjoint {True, False}

          article 5
            lstmt 5: implicit all_disjoint {Normal_VAT, Reverse_Charge_VAT,
                                            Full_VAT, Sales_Tax}
        end
        legalsec 2 "Rules"

          article 6 /* Determine VAT Identifier and adj.
                       for Payment discount */
            lstmt 6: vatProductPG(this) in VAT25
                      -> vatIdentifier(this) in VAT25 and
                         adjustForPaymentDiscount(this) in False

          article 7 /* Determine VAT Percentage */
            lstmt 7: vatProductPG(this) in VAT25
                      -> vatPercentage(this) = 25
            lstmt 8: exception_to {7} vatBusinessPG(this) in EXPORT
                       -> vatPercentage(this) = 0

          article 9 /* Determine sales and purchase accounts */
            lstmt 9: vatProductPG(this) in VAT25
                      -> salesVatAccount(this) = 5610 and
                         purchaseVatAccount(this) = 5630

          article 10 /* Determine VAT calculation type and reverse
                         charge VAT account */
            lstmt 10: vatBusinessPG(this) in EU and
                       vatProductPG(this) in VAT25
                       -> vatCalculationType(this) in
                                        Reverse_Charge_VAT and
                       reverseChargeVatAccount(this) = 5620

            lstmt 11: implicit Non-EU = U {EXPORT, NATIONAL}

            lstmt 12: vatBusinessPG(this) in Non-EU
                       -> vatCalculationType(this) in Normal_VAT
        end
      end
end
```

B.2 LLMBRR Encoding of VAT on Danish Services

```
model
  legaldoc "VAT on Services for Danish Companies"
    header
      enacted: "26:05:2005"
      importance-level: 3
    body
      legalsec 1 "The Main Rules"
        legalsec 1 "Determine Place of Delivery"
          article 1 /* The Main Rule */
            lstmt 1: this in Services and
                     sellerEstablishedIn(this) in Denmark
                     -> vatPlaceOfDelivery(this) in Denmark
        end
        legalsec 2 "Determine VAT Rate and Scheme"
          article 2 /* The Main Rule */
            lstmt 2: this in Services and
                     vatPlaceOfDelivery(this) in Denmark
                     -> vatPercentage(this) = 25 and
                        vatScheme(this) in Normal_VAT
        end
        legalsec 3 "Who calculates the VAT"
          article 3 /* The Main Rule */
            lstmt 3: this in Services and
                     vatPlaceOfDelivery(this) in Denmark
                     -> calculateVAT(this) in Seller
        end
      end
      legalsec 2 "Exceptions"
        legalsec 1 "Electronic Services"
          legalsec 1 "When the Seller is Danish"
            article 4
              lstmt 4: implicit ElectronicServices sub Services;
                       PrivateOrTaxablePerson = U {PrivatePerson, TaxablePerson};
                       all_disjoint {PrivatePerson, TaxablePerson}

              lstmt 5: exception_to {1} this in ElectronicServices and
                       buyerType(this) in PrivateOrTaxablePerson and
                       buyerCountry(this) in Denmark
                       -> vatPlaceOfDelivery(this) in Denmark and
```

```
                    calculateVAT(this) in Seller

            lstmt 6: implicit all_disjoint {Buyer, Seller};
                     Denmark sub EU;
                     all_disjoint {Denmark, NotDenmark}

            lstmt 7: exception_to {1} this in ElectronicServices and
                     buyerType(this) in TaxablePerson and
                     buyerCountry(this) in EU and
                     not buyerCountry(this) in Denmark
                     -> /* Hack for not on right hand side */
                        vatPlaceOfDelivery(this) in NotDenmark and
                        vatScheme(this) in Reverse_Charge_VAT and
                        calculateVAT(this) in Buyer

            lstmt 8: exception_to {1} this in ElectronicServices and
                     buyerType(this) in PrivatePerson and
                     buyerCountry(this) in EU and
                     not buyerCountry(this) in Denmark
                     -> vatPlaceOfDelivery(this) in Denmark and
                        calculateVAT(this) in Seller

            lstmt 9: implicit all_disjoint {Non-EU, EU}

            lstmt 10: exception_to {1} this in ElectronicServices and
                      buyerType(this) in PrivateOrTaxablePerson and
                      buyerCountry(this) in Non-EU
                      -> vatPlaceOfDelivery(this) in NotDenmark
        end
        legalsec 2 "When the Buyer is Danish"
          article 5
            lstmt 11: this in ElectronicServices and
                      sellerEstablished(this) in Denmark and
                      buyerType(this) in PrivateOrTaxablePerson and
                      buyerCountry(this) in Denmark
                      -> vatPlaceOfDelivery(this) in Denmark and
                         calculateVAT(this) in Seller

            lstmt 12: exception_to {3} this in ElectronicServices and
                      sellerEstablished(this) in EU and
                      not sellerEstablished(this) in Denmark and
                      buyerType(this) in TaxablePerson and
                      buyerCountry(this) in Denmark
                      -> vatPlaceOfDelivery(this) in Denmark and
                         calculateVAT(this) in Buyer

            lstmt 12: this in ElectronicServices and
                      sellerEstablished(this) in EU and
                      not sellerEstablished(this) in Denmark and
                      buyerType(this) in PrivatePerson and
                      buyerCountry(this) in Denmark
                      -> vatPlaceOfDelivery(this) in NotDenmark and
                         calculateVAT(this) in None

            lstmt 13: exception_to {3} this in ElectronicServices and
                      sellerEstablished(this) in Non-EU and
                      buyerType(this) in TaxablePerson and
                      buyerCountry(this) in Denmark
                      -> vatPlaceOfDelivery(this) in Denmark and
                         calculateVAT(this) in Buyer

            lstmt 14: this in ElectronicServices and
                      sellerEstablished(this) in Non-EU and
                      buyerType(this) in PrivatePerson and
                      buyerCountry(this) in Denmark
                      -> vatPlaceOfDelivery(this) in Denmark and
                         calculateVAT(this) in Seller
          end
        end
      end
end
```

C

E-mail correspondance

In this appendix we have included our mail correspondance (in Danish) with Kim Graversen. The mails have been stripped from personal comments, but have not been formatted or modified otherwise.

C.1 First Mail from Kim Graversen

```
Kim sent you a message.

--------------------
Re: SAP Guru?

Hej Morten <personal comments removed>

Du kan tro jeg ved noget om SAP's momsstyring  :)

Generelt set styres momstilgangen ud fra kundens landekode opsat på masterdata
(og der er en hel særlig "kunde"-definition her ang. sold-to, ship-to, bill-to
og payer...som jo kan være forskellige)

Ud fra den lokation hvor man som sælgende enhed leverer ydelsen fra (er det en fysisk
vare er det jo en decideret "fabrik" i sap termer og det er denne fabriks lokation
landemæssigt som gælder) bestemmes nu om der er tale om domestic sales eller export sales.
Man bestemmer dette i det der kaldes tilgangssekvenser og der ledes efter opfyldte
konditioner i prioriteret orden. Man sætter så en tabel op for eksempel efter nedenstånede
model (for export)

Receiver land - materiale momsindikator - kundemomsindikator- momskode
Timbuktu - full moms - full moms - salgmomskode 25%
Tyskland (EU) - full moms - full moms - salgsmomskode EU 0%
og der er så indikatorer på materiale/kunde som kan angive halv moms etc.
og dermed trigge en anden momskode

På domestic siden er det lig ovenstående men selvf. uden receiverland

Dette er en direkte relation mellem virksomheden og kunden (hvad enten det er
en btb kunde eller privat)... Men man kan også forestille sig en situation hvor man har et
centralt lager liggende et givet sted sted i europa og har et salgssted lokalt som sælger
til lokal kunde man hvor varen shippes direkte fra udlandsfabrik til privatkunde.

Dette håndteres ved at trigge 2 dokumenter:

1) salgsdokument mellem lokal salgssted og lokal kunde (hvis dk, så typisk 25% alm. dansk
```

```
salgsmoms)
2) Intercompany salgs(købs)dokument mellem lageret og salgsstedet. Dette gælder
ovenstående regler mellem om det er decideret export eller indenfor EU imellem virksomhedens
2 loaktioner. Hvis fx lager i UK, så trigges en salgsmoms fra England på EU 0% og tilsvarende
et købsmoms på den danske salgssted  på EU 0%...

Men dette er blot en særlig varient - i bund og grund er der som beskrevet ovenstående tale
om tilgangssekvenser hvor der ledes i prioriteret rækkefælge efter landedetermineringen
mellem kunde og virksomed og derefter hvordan konditionstabllen er sat op med de
informationer der findes på stamdata på kunde/materiale og salgsordren specifikt-

Håber det kan give lidt hjælp... <personal comments removed>

Ellers er der altid hjælpebeskriveser at finde på help.sap.com (uden www)
og det er åbent for alle !

Jeg har fundet et link som gerne skulle understøtte ovenstående  ;-)

http://help.sap.com/erp2005_ehp_03/helpdata/EN/e5/077d5c4acd11d182b90000e829fbfe/frameset.htm
--------------------
```

C.2 Second Mail from Kim Graversen

```
Hej Morten <personal comments removed> Jeg svarer med STORT nedenstående:

Hej Kim,
Så har jeg lige et par spørgsmål om moms i SAP.

Det du har forklaret ovenfor (og som giver fin mening) giver anledning til, at man
identificerer en "Tax Code", som styrer hvordan moms skal beregnes.

Jeg kan læse mig til nogle ting vedr. hvordan Tax Codes defineres og anvendes.
Så vidt jeg forstå gør man følgende, når man skal lave en ny Tax Code:

1. Vælg en "Country Key". Denne giver i nogle tilfælde andgang til standard
"Calculation Procedures". Hvis ikke skal man selv definere dem.

JA DER ER 1 PR. LAND - OG DE SKULLE ALLE VÆRE LEVERET MED SAP STANDARD - NOGLE
GANGE KAN MAN DOG FOR NOGLE AF DE NYE LANDE SELV SKULLE DEFINERE DEM (EX-JUGOSLAVIEN
OG EX-SOVJETLANDE) DA DE ER SÅ "NYE"

2. Calculation Procedures er en liste af de forskellige "Tax Types" som findes
for den pågældende Country Key.

IKKE HELT, SNARERE OMVENDT - TIL EN CALCULATION PROCEDURE ER DER ET ANTAL TAX
TYPES PÅHÆFTET

3. Hver Tax Type indeholder information om: base amount, calculation type, side of
account og tax expense.

NEJ IKKE DET HELE - EN TAX TYPE SIGER FAKTISK KUN NOGET OMKRING POSTERINGEN (KONTOEN
(G/L ACCOUNT)), OM DET ER UDGÅENDE/INDGÅENDE MOMS, OG SÅ EN HENVISNING TIL EN
VALIDERINGSTABEL HVOR LANDET OG KODEN ER SAT SAMMEN

MEN DET ER I SAMMENHÆNG MED CALCULATION PROCEDUREN DE ANDRE TING OPSTÅR. CALCULATION
PROCEDUREN ER ET STANDARDSKEMA DER ANVENDES FOR HELE LANDET, OG MAN ANGIVER SÅ FOR
HVER MOMSKODE, HVILKEN TAX TYPES DER TRIGGES MED HVILKEN PROCENT. NEDENSTÅENDE SKÆRMDUMP
ER FX FOR SALGSMOMSKODE S1 FOR DK. BASE AMOUNT ER ALTSÅ SAT OP PÅ CALCULATION PROCEDURE

Og så vidt så godt.

A. Det jeg ikke forstår er, hvor man angiver, hvilken G/L account, som VAT skal posteres til
for de enkelte Tax Types.
```

DET GØR MAN SÅ I EN TABEL DER TAGER FAT I KONTOPLAN OG OVENSTÅENDE TAX TYPE (mws OVENSTÅENDE)
OG EVT. TAX CODE (DET KAN MAN SELV BESTEMME OM MAN VIL HAVE KONTOEN DETERMINERET PR. TAX CODE,
ELLER SOM NORMALT AT MAN ANVENDER SAMME KONTO FOR HVER TAX TYPE, SÅ BEHØVER MAN KUN TABEL MED
LAND, TAX TYPE) - HER PÅHÆFTES g/l ACCOUNT. DA DET ER KONTOPLAN OG IKKE LAND DER ANVENDES I
DENNE TABEL, KAN MAN ALTSÅ GODT SÆTTE DETTE OP UAFHÆNGIGT AF HVILKE LANDE MAN OPERERER I -
MEN FÅ POSTERINGEN GENERELT PÅ SIN FIRMAKONTOPLAN

B. Endivdere ved jeg, at der i Tyskland (modsat Danmark) er særlige regler for hvornår (på
posteringstidspunktet hhv. faktureringstidspunktet) man skal postere moms. Understøtter SAP
det, og i givet fald er det så noget man kan indstille?

DET SKAL JEG SPØRGE EN KOLLEGA OM - FOR SELVFØLGELIG UNDERSTØTTER VI DET I SAP MEN JEG ER
IKKE HELT KLAR OVER HVAD DIN FORSKEL ER I SPØRGSMÅLET. NÅR MAN LAVER EN UDGÅENDE FAKTURA I
SAP - SÅ FORETAGES KONTERINGEN UMIDDELBART - OGSÅ MOMSLINIEN - FOR ALLE LANDE. DER ER INGEN
TIDSDIFFERENCE HER

C. Endelig kan det være du kan opklare om VAT beregnes pr. sales line, pr. sales order (invoice)
eller begge dele.
DET FORETAGES PR. SALGSLINIE, IDET DER I SALGSMODULET ER EN INDIVIDUEL PRISKALKULERING (INKL MOMS)
PR. LINIE (DENNE SUMMES SÅ TIL HEADER, SÅ DER ER EN SAMLET POSTERINGSOVERSIGT). MAN KAN NU I SAPS
SENESTE VERSION SELV BESTEMME HVORVIDT MOMSEN SKAL POSTERES PR. ET ELLER ANDET OBJEKT (FX PR.
KONTERINGSOBJEKT, PROFIT CENTER) ELLER SOM HIDTIG BLOT LÆGGES SOM EN (SUM)LINIE PR. UDSTEDT
FAKTURA - SÅ DET ER ET LIDT BÅDE OG SPØRGSMÅL J SPLIT ER ET NYT BEGREB I SAPS NEW GENERAL LEDGER
SOM KALDES "DOCUMENT SPLITTING" OG ER ALTSÅ SPRITNYT I SAPS 2 SENESTE ERP-VERSIONER

D. En anden ting er så, hvordan selve beregningerne udføres. Fx er der nogle lande, hvor moms
først skal posteres, når kunden betaler fakturaen (man taler om unrealized VAT) og dette findes
der så forskellige måder at gøre på, fx hvis kunden betaler over flere omgange at gøre det
proportionalt således at en lige stor del af totalen uden moms og selve momsen posteres hver
gang en betaling ankommer. Dette leder mig så frem til at efterlyse en "event-model" for,
hvordan SAP håndtere moms, eller sagt på en anden måde, hvad er det for nogle hændelser, som
trigger momsaktiviteter i SAP. Jeg kan oplyse at moms "event-modellen" in Microsoft NAV ser
nogenlunde ud som følger:
DET ER JEG IKKE HELT KLAR OVER HVORDAN SPECIFIKT HÅNDTERES - JEG GÅR UD FRA DET ER DET DER
KALDES "WITHHOLDING TAX" I SAP. JEG KAN GODT SPØRGE OM DER ER NOGEN DER HAR PRØVET DET FRA
KUNDENS SIDE SOM ER DET DU SPØRGER OM HER (EXTENDED WITHHOLDING TAX I SAP) MEN JEG TROR DET NU
IKKE. JEG KAN KUN GIVE DIG LINK HER TIL DEN DOKUMENTATION DER FINDES OMKRING EMNET, MEN JEG HAR
IKKE PRØVET DET I PRAKSIS...

http://help.sap.com/saphelp_erp60_sp/helpdata/en/e5/078d0b4acd11d182b90000e829fbfe/frameset.htm

VÆLG TAXES->WITHHOLDING TAX->EXTENDED FX

<personal comments removed>

C.3 Third Mail from Kim Graversen

Hej Kim,

<personal comments removed>
1. Du nævner nedenfor, at den G/L acct, som VAT skal posteres til
bestemmes bl.a. på baggrund af kontoplan. Jeg forstår det således, at
SAP supporterer brug af flere kontoplaner.
Jeg har nu følgende spørgsmål:
A. Hvordan finder man ud af, hvilken kontoplan man skal bruge (eller
sagt på en anden måde, når man laver et salg, hvor kommer informationen
om kontoplan så fra)?

MAN BINDER HELE SIN ORGANISATIONSPLAN SAMMEN I EN RÆKKE ASSIGNMENT TABELLER. KONKRET FINDES DENNE
VED AT NÅR DU LAVER SALGET VÆLGER DU EN SALGSORGANISATION. DENNE SALGSORGANISATION TILHØRER EN
FIRMAKODE SOM IGEN TILHØRER EN KONTOPLAN. RELATIONEN ER 1 TIL MANGE HER. EN KONTOPLAN KAN HAVE FLERE
FIRMAKODER SOM KAN HAVE FLERE SALGSORGANISATIONER - MEN FRA SALGSORGANISATIONEN ER VEJEN "OP" ENTYDIG

B. Er acct.nr. unikke på tværs af kontoplaner (kan en acct. fx figurere

```
på flere kontoplaner)?

DET KAN DEN FOR SÅ VIDT GODT - ALTSÅ BÆRE SAMME NUMMER - MEN DET HAR INGEN BETYDNING/RELATION. EN ACCT
ER UNIK INDENFOR SIN KONTOPLAN. HAR DU EN ANDEN KONTOPLAN SOM HAR DEN SAMME KONTO - KAN DE VÆRE VIDT
FORSKELLIGE I DERES OPSÆTNING/NATUR (DEN ENE KAN VÆRE EN OMKOSTNINGSKONTO OG DEN ANDEN SOM HER
EKSEMPELVIS EN MOMSKONTO - DET ER ET TILFÆLDE - SÅ KONTOEN ER UNIK INDENFOR SIN KONTOPLAN. DU KAN SIGE
AT BÅDE SELVE KONTOPLANEN OG KONTOEN DANNER NØGLEN

2. Kan man sige, at der i SAP eksisterer en mængde af Tax Types, og at
hver Calculation Procedure (som er unik pr. land) er associeret med en
delmængde heraf? Eller sagt på en anden måde:
Hver Calc.proc er påhæftet et antal Tax Types og en givet Tax Type A kan
være påhæftet flere Calc.proc.

JA LIGE PRÆCIS

<personal comments removed>
```

C.4 Fourth Mail from Kim Graversen

```
Hej Kim,

<personal comments removed>

Men her får du lige de to seneste spørgsmål:

1. På det første billede nedenfor er "Tax type" navnet på et felt og
"Tax Type" navnet på en kolonne. Hvilken rolle spiller "Tax type"?
Hvilke værdier kan den antage?

JA, DET ER DESVÆRRE EN OVERSÆTTELSESBØF - DET SER MAN DESVÆRRE MANGE GANGE.
DET ER JO ET TYSK SYSTEM, OG DET ER IKKE ALTID OVERSÆTTERNE HAR IT-VIDEN....

TAX TYPEN SOM ER DEN VI HELE TIDEN HAR SNAKKET OM (MÅSKE SAMMEN MED TAX
CODEN) - DET ER DEN I KOLONNEN, SOM KAN ANTAGE NOGLE VÆRDIER Á LA DEM VI HAR
NÆVNT HER I DK'S KALKULATIONSSKEMA (MWS, VST, NVV, ESA, ESE). DE STYRER SOM
NÆVNT PRIMÆRT KONTERINGEN TIL GL ACCOUNT TAX TYPE SOM FELT, BURDE HAVE HEDDET
NOGET ANDET, NU NÅR TYPEN ALLEREDE ER "OPTAGET", MEN DET GAD MAN ALTSÅ IKKE
LEDE EFTER EN ANDEN OVERSÆTTELSE - FELTET TAX TYPE ER AFLEDT FOR TABELLEN MED
TAX CODEN (S1 I BILLEDET), HVOR MAN ANGIVER OM TAX TYPEN (SOM FELT) SKAL VÆRE
IND- ELLER UDGÅENDE MOMS (OG DER ER KUN DE 2 MULIGHEDER HER - a FOR UDGÅENDE)

2. Er kolonnen "Tax Type" i nederste del af første billede nedenfor en
liste af alle de Tax Types, som findes for TAXDK? Kan jeg endvidere
forstå det sådan, at "Tax Type" er et "pretty name" for "Cond. Type" og
at de 4 andre søjler er parametre, som de givne Tax Types er
instansieret med for Tax Code S1 (altså hvis man nu skiftede til
en anden Tax Code under TAXDK så ville det være de samme "Tax Type"s
men med andre parametre i de omtalte 4 søjler).

NÆSTEN... MED EN ANDEN MOMSKODE, FX K1 FOR 25 % KØKSMOMS VILLE KODEN
ÆNDRE SIG I BILLEDET SELVFØLGELIG, OG TAX TYPE SOM FELT ÆNDRE SIG TIL
INDGÅENDE MOMS (AFLEDT AF K1-KODEN) OG DERNÆST VILLE TABELLEN ÆNDRE
SIG FRA AT VÆRE 25% I LINIEN MED VST SOM TAX TYPE  I STEDET FOR MWS -
ALLE ANDRE UÆNDRET.

CONDITION TYPE ER HÆGTET PÅ TAX TYPE OG ANVENDES I EN STYRETABEL TIL
AT VALIDERE DE ENKELTE KODER PR. LAND. DET VIRKER LIDT MEGET MED ALLE
DISSE STYRETABALLER DER HVER FOR SIG NÆSTEN KUN STYRER EN ENKELT TING,
MEN DET ER JO FOR AT MAN KAN KOMBINERE PÅ KRYDS OG  TVÆRS MELLEM LANDE,
SKEMAER OG TAX TYPES UDEN AT SKULLE OPRETTE OVERFLØDIGE ENTRIES.

DERFOR ER CONDITION TYPE IKKE BLOT ET PRETTY NAME HELT TEORETISK SET,
MEN I PRAKSIS HAR JEG ALDRIG SET AT FX MWS ER KNYTTET TIL ANDET END
```

```
MWAS OG AT VST ER KNYTTET TIL ANDET END MWVS... SÅ DET ER MERE I TEORIEN
DET KAN LADE SIG GØRE AT OPRETTE EGNE KONDITIONSTYPER TIL AT HÆGTE PÅ
TAX TYPEN

3. Hvad betyder Level?

DET ER BLOT EN COUNTER/TABELRÆKKE - MEN DET ER VIGTIGT NÅR MAN SER PÅ
"FROM LEVEL" FOR ØVERST STÅR BASE AMOUNT I LINIE 100, UD FRA
KONDITIONSTYPEN (basb) VED SAP AT DETTE ER GRUNDBELØBET I POSTERINGEN
(FX PÅ SALGSFAKTURAEN). NÅR MAN DEREFTER SKAL BEREGNE 25% SKAL MAN JO
FORTÆLLE "AF HVAD" OG DET GØR MAN SÅ VED AT REFERERE TIL LINIE 100 I
"FROM LEVEL", SÅ SKAL MAN ALTSÅ TAGE 25% AF LINIE 100 OG POSTERE SOM MOMS.

DER ER ANDRE MOMSKODER SOM BEREGNES MERE SOFISTIKERET, HVOR DET IKKE ER
AF BASISBELØBET, MEN FØRST BEREGNES EN GRUNDMOMS, OG DERNÆST EN
EKSTRAMOMS SOM ER EN PROCENTSATS AF DEN FUNDNE MOMS, OG IKKE AF
BASISBELØBET IGEN. SÅ VILLE MAN ALTSÅ BLOT REFERERE TIL EN ANDEN RÆKKE
I SKEMAET I "FROM LEVEL"

<personal comments removed>
```

References

1. Peter Blume. *Juridisk Metodelære.* Jurist- og Økonomiforbundets forlag, 4 edition, 2006.
2. Configit Software. Configit product modeler, version 4.1. Limited edition demo available from `http://www.configit.com`.
3. Sean Bechhofer et al. Owl web ontology language reference.
4. Volker Haarslev and Ralf Möller, editors. *Proceedings of the 2004 International Workshop on Description Logics*, volume 104. CEUR-WS, 2004.
5. E. Heinäluoma. Council Directive 2006/112/EC of 28 November 2006 on the common system of value added tax. *Official Journal of the European Union (English edition)*, Volume 49 11(L 347), December 2006.
6. Ian Horrocks and Peter F. Patel-Schneider. Reducing owl entailment to description logic satisfiability. *Lecture Notes in Computer Science*, 2870/2003:17–29, September 2003.
7. P. Leith. Fundamental errors in legal logic programming. *Comput. J.*, 29(6):545–552, 1986.
8. Joel J. Lerner. *Bookkeeping and Accounting.* McGraw-Hill Education, 2004. Isbn: 0-07-142240-4.
9. A. Rector R. Stevens M. Horridge, H. Knublauch and C. Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf, 2004.
10. Momsvejledningen 2008-3. http://www.skat.dk/SKAT.aspx?oId=95819.
11. Morten Ib Nielsen. A System of VAT Rules. Mail mortenib@diku.dk to obtain a copy, 2007.
12. Morten Ib Nielsen. Tutorial on Modeling VAT Rules Using OWL-DL. Mail mortenib@diku.dk to obtain a copy, 2007.
13. Morten Ib Nielsen, Jakob Grue Simonsen, and Ken Friis Larsen. Logical models for value-added tax legislation. Submitted to LPAR 2008.
14. Told og Skattestyrelsen, editor. *Moms af ydelser.* ToldSkat, Østbanegade 123 DK-2100 København Ø, 2005. Isbn: 87-7552-699-9.
15. Henry Prakken. *Logical tools for modelling legal argument : a study of defeasible reasoning in law.* Kluwer Academic Publishers, 1997.
16. Sap library: Taxes (fi-ap/ar). http://help.sap.com/saphelp_erp60_sp/helpdata /en/e1/8e51341a06084de10000009b38f83b/frameset.htm.

17. M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, 1986.
18. Skatteministeriet. Lbk nr 966 af 14/10/2005.
19. York Sure and Óscar Corcho, editors. *EON2003, Evaluation of Ontology-based Tools, Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools held at the 2nd International Semantic Web Conference ISWC 2003, 20th October 2003 (Workshop day), Sundial Resort, Sanibel Island, Florida, USA*, volume 87 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.
20. F.P. Coenen T.J.M. Bench-Capon. Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, 1(1):65–86, March 1992.
21. Uk: Value added tax act 1994.
22. Wil van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In Frank Leymann, Wolfgang Reisig, Satish R. Thatte, and Wil van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. <http://drops.dagstuhl.de/opus/volltexte/2006/829> [date of citation: 2006-01-01].
23. Jerry J. Weygandt, Donald E. Kieso, and Paul D. Kimmel. *Financial Accounting, with Annual Report*. Wiley, 2004.

Tutorial on Modeling VAT Rules Using OWL-DL

Morten Ib Nielsen[*], Jakob Grue Simonsen, and Ken Friis Larsen[*]

Department of Computer Science, University of Copenhagen, Denmark
{mortenib|simonsen|kflarsen}@diku.dk

Abstract. This paper reports on *work in progress.* We present a methodology for constructing an OWL-DL model of a subset of Danish VAT rules. It is our intention that domain experts without training in formal modeling or computer science should be able to create and maintain the model using our methodology. In an ERP setting such a model could reduce the Total Cost of Ownership (TCO) and increase the quality of the system. We have selected OWL-DL because we believe that description logic is well suited for modeling VAT rules due to the decidability of important inference problems that are key to the way we plan to use the model and because OWL-DL is relatively intuitive to use.

1 Introduction

Imagine an ERP system where domain experts can create and implement changes in e.g. VAT rules without the help of programmers. The benefits would be shorter development time and fewer mistakes due to misinterpretation of specifications, which lead to reduced TCO and increased quality of the software. On a coarse-grained scale such a system consists of three parts: A model of the rules, a tool to edit the model and the core ERP system using the model. In this paper we focus on the first part - the model. A priori two requirements exist. First the modeling language must be strong enough to express the rules in question and second it must be easy to use without training in formal modeling or computer science. In a more general setting the model can be used as a VAT knowledge system, which external programs can query through an interface. In the long run we envision that authorities such as SKAT (Danish tax administration) can provide online access to the model (e.g., using web services such that applications always use the newest version of the model).

In this paper we describe the methodology we have used to develop a model of a subset of Danish VAT rules using the general purpose Web Ontology Language (OWL) editor Protégé-OWL[1] and we report on our experiences in doing so. We selected a subset of Danish VAT rules consisting of flat VAT (25%) plus a set of exceptions where goods and services are free of VAT, chosen because they seem representative. Further the rules are accessible to us by way of an official guideline by the Danish tax administration. Our study is focusing on

[*] Supported by the Danish National Advanced Technology Foundation.

[1] `http://protege.stanford.edu/overview/protege-owl.html`.

the feasibility of using OWL to model VAT rules and not on the usability of the Protégé-OWL tool itself. By feasibility we mean how easy or difficult it is (for a human) to express and understand VAT rules in OWL, in particular this does not cover issues such as modularization. The methodology presented here is inspired by [3]. Readers of this guide are assumed to have user experience with Protégé-OWL corresponding to [2] but not of computer science nor of modeling in general.

1.1 Motivation

One of the overall goals of the strategic research project 3gERP is to reduce the TCO of Enterprise Resource Planning (ERP) systems. We believe that a VAT model helps to this end in two ways. First we envision that domain experts create and update the model thus eliminating a layer of interpretation (the programmer) where errors can be introduced. Second a VAT model can change handling of VAT from being a customization task into being a configuration task, meaning that no code needs to be changed when the model is updated.

VAT and legal rules in general deal with frequent transactions between legal entities. Transactions are typically triggered when certain conditions are fulfilled and therefore dynamic checks on these conditions are needed. The idea is to use the model to automatically infer what actions should be taken based on the conditions. In the case of VAT rules we can ask the model whether a delivery is subject to VAT or not based on the information we know about the delivery. The answer from the model will be *Yes*, *No* or *Maybe*[2] and can be used to trigger an appropriate transaction. In a broader perspective the model is supposed to work as a VAT knowledge system that given a context and a question can tell other systems what to do (e.g., guide accounting systems and if required indicate that authorities should be contacted etc.).

1.2 Roadmap

The remainder of this paper is structured as follows. In Section 2 we give a short account of description logic and OWL. In Section 3, 4 and 5 we present our methodology by giving examples. Finally we outline future work in Section 6 and we conclude in Section 7.

2 Description Logic and OWL

In this section we give a short introduction to description logic (DL) and OWL. This introduction can be skipped, if you are already familiar with the concepts. Description logics are knowledge representation languages that can be used to structure terminological knowledge in knowledge systems which are formally well-understood. A knowledge system typically consists of a knowledge base

[2] In the case where insufficient information is provided in order to answer the question.

together with a reasoning service. The knowledge base is often split into a set of concept axioms the *TBox*, a set of assertions the *Abox* and a *Role hierarchy*. These constitute the *explicit* knowledge in the knowledge system. The reasoning service is a program that can check the consistency of the knowledge base and make implicit knowledge explicit (e.g., decide equivalence of concepts). Since the reasoning service is a pluggable component knowledge systems separate the technical task of reasoning from the problem of constructing the knowledge base.

2.1 OWL

OWL which is short for Web Ontology Language is an ontology language designed to be compatible with the World Wide Web and the Semantic Web. The most important abstraction in OWL is concept axioms (called classes). Each class has a list of necessary conditions and zero or more equivalent lists of necessary and sufficient conditions [2]. A list of necessary conditions is a list of conditions that every member of the class must satisfy. In the same way a list of necessary and sufficient conditions is a list of conditions that must be satisfied by every member of the class and if satisfied guarantees membership in the class. OWL is based on XML, RDF and RDF-S and can be used to represent information in a way that is more accessible to applications than traditional web pages. In addition OWL has a formal semantics, which enables logic reasoning. OWL comes in three variants: OWL-Lite $\subseteq$ OWL-DL $\subseteq$ OWL-Full of increasing expressive power. The variants OWL-Lite and OWL-DL are based on the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ respectively [1], which guarantees that important inference problems such as satisfiability and subsumption are decidable. Since OWL is XML based we need an editor to create OWL ontologies. We have used the general purpose OWL editor Protégé developed by Stanford Medical Informatics at the Stanford University School of Medicine.

3 VAT Exemption 1: Sales Outside EU

Our methodology is aimed at modeling VAT rules as described in guidelines instead of the raw law text itself. This choice was made because guidelines are more accessible to us, and because these are the rules that small companies adhere to in practice. Further the investigation of the feasibility of using OWL to model VAT rules concerns the ease with which rules can be formalized and not so much from where the rules are extracted[3]. In what follows we refer to the guideline as the *legal source*.

In order to ease reading we have used the word *concept* only when we speak about the legal source. The corresponding concept in the model (OWL) is called a *class*. A concept in the legal source is modeled as one or more classes in the model.

Here we present the steps we took in order to make our model of Danish VAT rules.

[3] Since we have used the official guidelines by SKAT (Danish tax administration) we believe that the content of the guidelines is in accordance with the law.

3.1 Pre-modeling

1. Download Protégé-OWL from `http://protege.stanford.edu/download/release/full/` and install. Make sure you can start Protégé in OWL-mode (logic view). When started and if you select the *Class* tab it should look like Figure 1.

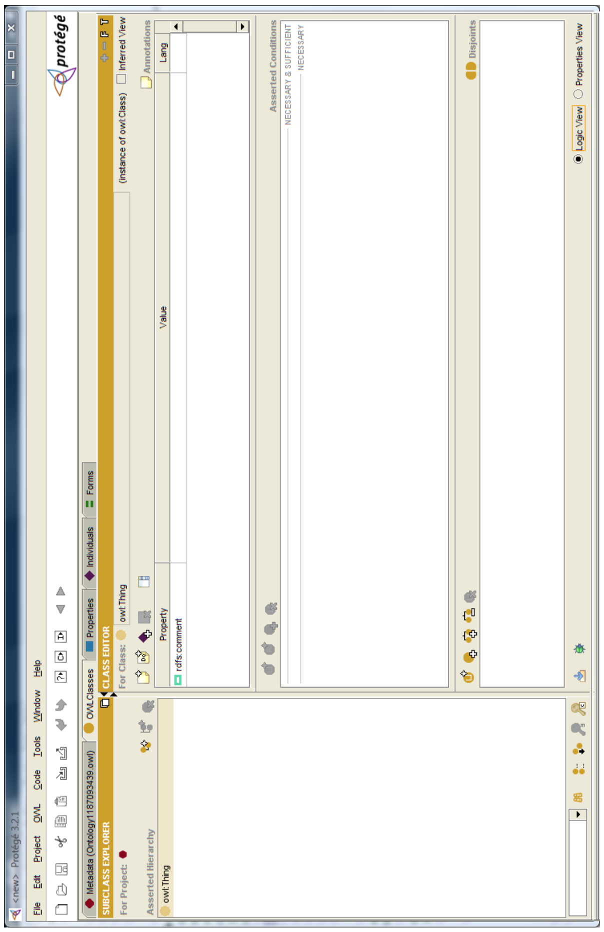


Fig. 1. Protégé-OWL class-tab, logic view.

2. Download [2] and read it. This is **important** because many of the constructions we use are explained herein.

3.2 Modeling

First you must decide which legal source(s) you want to model.

In our case we used the official guideline *Moms - fakturering, regnskab mv, E nr. 27, Version 5.2 digital, 19. januar 2005.*

Overall Framework Modeling should start with a read through of the legal source. Based on this general (to be refined later) classes such as `Location`, `Goods`, `Services` and `FreeOfVAT` together with attributes such as `hasDeliveryType` and `hasSalesPrice` can be created as subclasses of the built-in top-level class `owl:Thing`. Usually attributes are over finite domains and if so we use value partitions to model them as described in [2, p. 73-76][4]. If the domain is not finite we use data type properties instead. Deciding on the overall framework helps to structure the capturing of rules in a homogeneous way and enables working in parallel (which can be needed if the legal source is large). After our read through of the legal source we arrived at the overall framework in Figure 2.

`Naming Convention.` All classes, properties, individuals etc. should be given names picked from or inspired of the legal source. All names should be in the same language as the legal source (in our case Danish). Using the naming convention supported by Protégé-OWL class and individual names should be written in Pascal Notation (e.g., *InternationalOrganization* not *internationalOrganization* or *International_Organization*), while property names are written in Camel Hump Notation (e.g., *someProperty*). Typically a property is used to assign an attribute to a class. In this case we prefix the name of the property with a verb describing the kind of relation the class has along that property (e.g., *hasNumberOfSides* or *isFragile*).

Rule Modeling - Step I Having modeled the overall framework it is time to go through the legal source one section at a time looking for rules that should be modeled. Here we give an elaborate description of how to model a single rule from the legal source starting from the overall framework in Figure 2. In Section 4 and 5 we give a brief description of how to model other rules. Together the modeling of these rules cover all the constructions we have used in our VAT model. Since our legal source is in Danish we present the rules in their original Danish phrasing together with a translation into English. Now let us consider the rule shown in Table 1.

Since our model is only a prototype we make a slight simplification and assume that the rule applies to *all* services. With this simplification we can identify the necessary and sufficient conditions for application of the rule. These are shown in Table 2. In order to model the necessary and sufficient conditions in Table 2 we must add some attributes to `VarerOgYdelser`. The first and second condition in Table 2 tell us that we must be able to model that goods and services are sold[5]. We do that by adding an attribute to the class `VarerOgYdelser` (translates into `GoodsAndServices`) which already exists in our overall framework. Attributes are modeled using functional properties. In accordance with

[4] An exception is the domain of truth values, which is built-in as a data type.

[5] Instead of being sold goods can also be used as e.g. a trade sample. See [4, p. 8-9] for other examples.

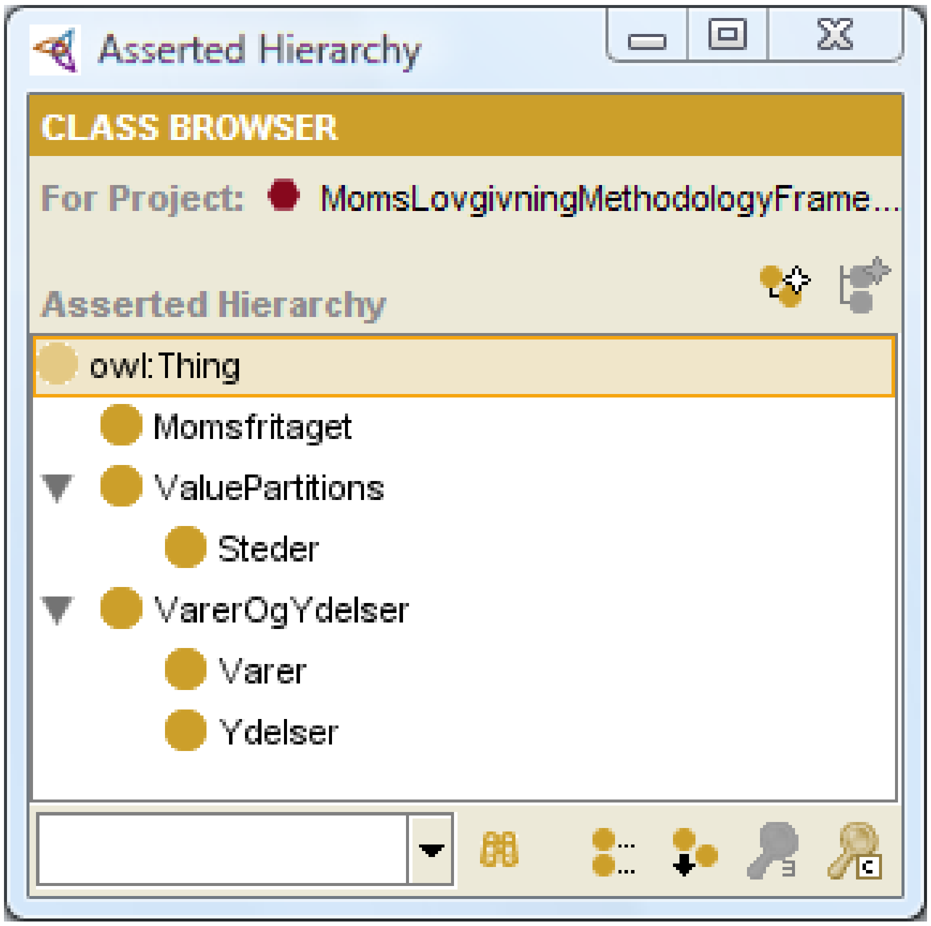


Fig. 2. Overall framework.

our naming convention we select the name `harLeveranceType` (translates into `hasDeliveryType`). Since there is a finite number of delivery types we model this attribute as a *value partition* (i.e., an enumeration). Value partitions can be created using a built-in wizard[6]. Just as in [2] we store value partitions as subclasses of the class `ValuePartitions`. The reason plain enumerations are not used is that they cannot be sub-partitioned. Using value partitions we retain the possibility of further refining the concepts the value partitions model.

Remark 1. Technically enumerations are constructed by defining a class in terms of a finite set of individuals plus a functional property that has this class as its range. Since individuals are atoms they cannot be subdivided. On the other hand

[6] Menu▶Tools▶Patterns▶Value Partition....

Salg til lande uden for EU (3. lande). Du skal ikke beregne moms af varer, du leverer til steder udenfor EU eller til Færøerne og Grønland. Det samme gælder normalt også for ydelser, men du skal dog opkræve moms af visse ydelser.

[4, p. 9]

And translated into English:

Sales outside EU (3rd countries). No VAT should be added to goods delivered to destinations outside the European Union, or to the Faroe Islands or Greenland. This fact ordinarily also applies to services, but VAT should be added to certain services.

Translated from [4, p. 9]

Table 1. Extract from the legal source and its translation into English.

– The rule concerns sales.
– The rule concerns both goods and services.
– The place of delivery must be outside the European Union, or the Faroe Islands or Greenland.

Table 2. Necessary & sufficient conditions for application of the rule in Table 1.

a value partition is defined using a functional property having as its range a class defined as the union of its subclasses all of which are distinct. These subclasses can (because they are classes) be partitioned into more subclasses if needed.

Having created the value partition `harLeveranceType` which can have `Salg` (translates into `Sale`) as a value we need to add it as an attribute to the class `VarerOgYdelser`. This is done by adding to the *necessary conditions* an existential quantification over the corresponding property having the value partition (or data type in case of data type attribute) as its range. Thus we add ∃ `harLeveranceType some LeveranceType` to `VarerOgYdelser`. The third condition suggests that we must be able to model that goods and services have a place of delivery. A read through of the legal source tells us that only three places are needed namely *Denmark*, *EU* and *non-EU*. Thus this attribute, which we name `harLeveranceSted` (translates into `hasPlaceOfDelivery`) must be modeled as a value partition.

Having modeled these attributes the class `VarerOgYdelser` looks as shown in Figure 3.

Rule Modeling - Step II Now we are ready to model the rule itself. Since the rule describes a situation where you do not have to pay VAT we model it as a subclass of `Momsfritaget` (translates into `FreeOfVAT`). Following our naming convention we name the class `MomsfritagetSalgAfVarerOgYdelserTilIkke-EU`

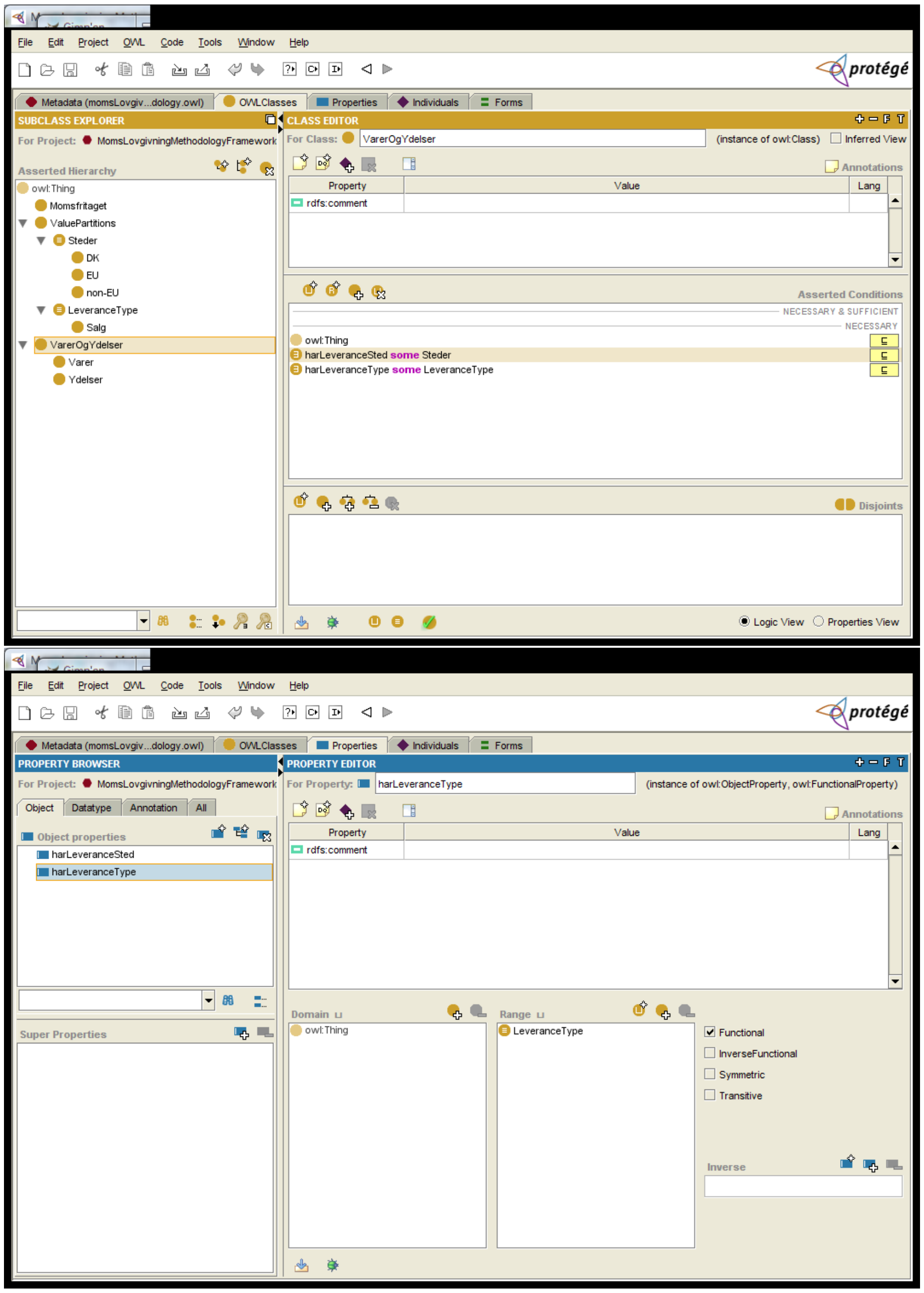
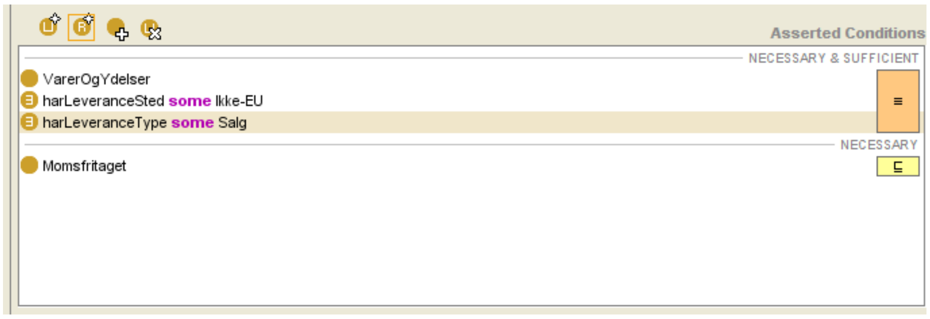


Fig. 3. Class and property view after adding attributes.

(translates into `VATFreeSalesOfGoodsAndServicesInNon-EU`). Then we add a textual description of the rule and a reference to where in the legal source the rule can be found in `rdfs:comment` field. Next we must specify *necessary and sufficient* conditions on membership in `MomsfritagetSalgAfVarerOgYdelserTilIkke-EU`. It is important to remember that if a class has two sets of necessary and sufficient conditions then they must imply each other, see [2, p. 98]. Based on the necessary and sufficient conditions captured in Table 2 we add the following necessary and sufficient conditions to `MomsfritagetSalgAfVarerOgYdelserTilIkke-EU`:

- `VarerOgYdelser`
- ∃ `harLeveranceSted some Ikke-EU`
- ∃ `harLeveranceType some Salg`

The result is shown in Figure 4.



VarerOgYdelser
harLeveranceSted some Ikke-EU
harLeveranceType some Salg
Momsfritaget

Fig. 4. Asserted Conditions of our model of the legal rule in Table 1.

4 VAT Exemption 2: Sales to Embassies

From this section and onwards we will not mention when to add references to the legal source in `rdfs:comment` fields of classes and properties. The rule of thumb is that this should always be done. Now let us consider the rule in Table 3. We identify the necessary and sufficient conditions for application of the rule. These are shown in Table 4.

4.1 Rule Modeling - Step I

We are already able to model that the rule concerns sale and that the place of delivery must be in EU, but we cannot model the specific service transportation yet. Therefore we must add it to our model. Since it is a service it should be modeled as a subclass of `Services`. We name the class modeling the service transportation `Transport` (translates into `Transportation`). Now we can model

> *Salg til ambassader. Du skal ikke beregne moms af varer og transportydelser, som du leverer til ambassader og internationale organisationer i andre EU-lande.*
>
> [4, p. 9]

And translated into English:

> *Sales to embassies. VAT should not be added to goods and transport services delivered to embassies and international organizations in countries within the European Union.*
>
> Translated from [4, p. 9]

Table 3. Extract from the legal source and its translation into English.

- The rule concerns sales.
- The rule concerns goods and transport services.
- The place of delivery must be in the European Union.
- The buyer must be an embassy or an international organization.

Table 4. Necessary & Sufficient conditions for application of the rule in Table 3.

that something belongs to the set of goods and transport services by requiring membership of `Varer` $\sqcup$ `Transport`. Finally we must be able to model that the buyer is an embassy or an international organization. Since there are only finitely many different kinds of buyers we model this as a value partition, and because this attribute applies to both `Varer` and `Transport` we add it to their most specific common super-class which is `VarerOgYdelser`. We name this attribute `harKøberType` (translates into `hasKindOfBuyer`). After having done all this the model looks as shown in Figure 5.

4.2 Rule Modeling - Step II

Having added all the necessary classes and attributes to the model we are ready to model the rule itself. Since the rule describes a situation where you do not have to pay VAT we model it as a subclass of `Momsfritaget`. Following our naming convention we name the class `MomsfritagetSalgTilAmbassaderOgInternation-aleOrganisationerIEU` (translates into `VATFreeSalesToEmbassiesAndInter-nationalOrganizationsInEU`). Based on the necessary and sufficient conditions captured in Table 4 we add the following necessary and sufficient conditions to `MomsfritagetSalgTilAmbassaderOgInternationaleOrganisationerIEU`:

- `harLeveranceType some Salg`
- `Varer` $\sqcup$ `Transport`
- `harLeveranceSted some EU`
- `harKøberType some AmbassadeOgPersonaleMedDiplomatiskeRettigheder`

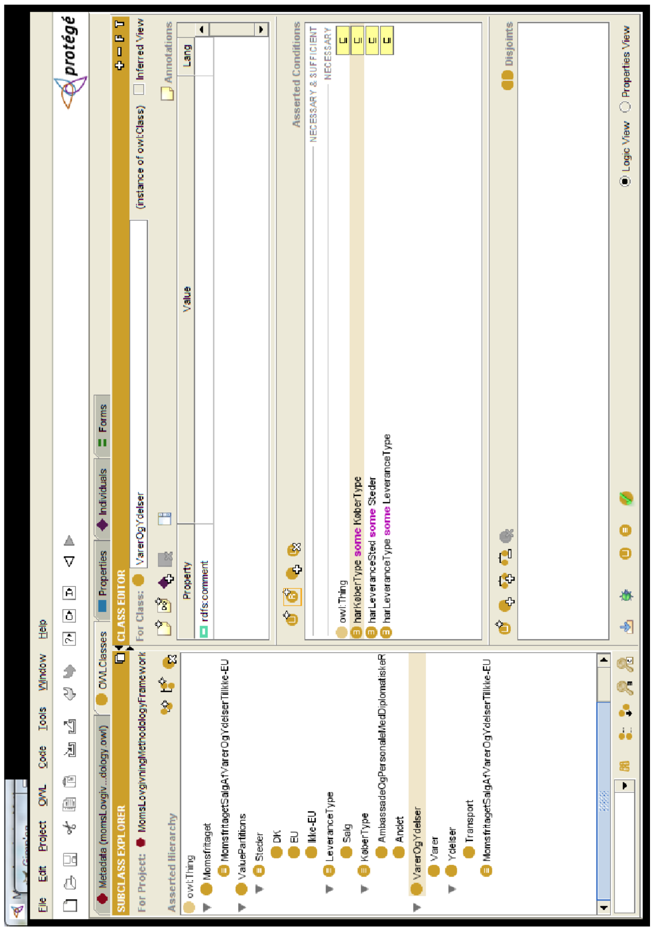
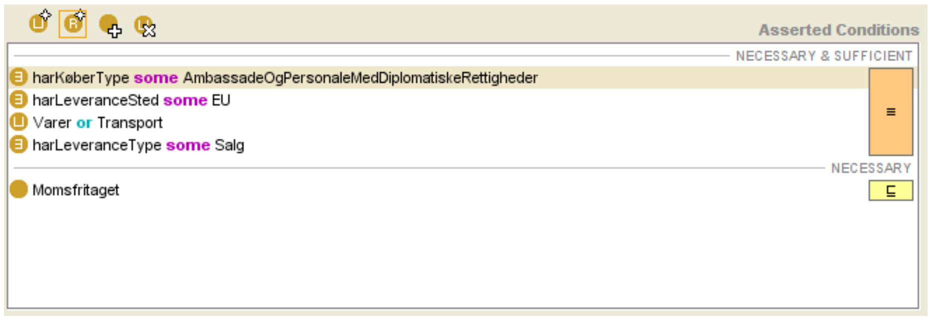


Fig. 5. The model after adding classes and attributes as described in Section 4.1.

The result is shown in Figure 6.



harKøberType **some** AmbassadeOgPersonaleMedDiplomatiskeRettigheder
harLeveranceSted **some** EU
Varer **or** Transport
harLeveranceType **some** Salg

Momsfritaget

Fig. 6. Asserted Conditions of our model of the legal rule in Table 3.

5 VAT Exemption 3: Sales in Other EU Countries

In this section we consider one final rule, the rule in Table 5. We identify the

> *Salg til andre EU-lande. Du skal ikke beregne dansk moms, når du sælger varer til momsregistrerede virksomheder i andre EU-lande. Du skal derfor sørge for at få virksomhedens momsnummer.*
>
> [4, p. 8]

And translated into English:

> *Sales in other EU countries. No VAT should be added to goods delivered to companies in other EU countries, provided that the companies are registered for VAT. In this case you must acquire the VAT registration number of the company.*
>
> Translated from [4, p. 8]

Table 5. Extract from the legal source and its translation into English.

necessary and sufficient conditions for application of the rule. These are shown in Table 6.

5.1 Rule Modeling - Step I

We are already able to model that the rule concerns sale of goods delivered inside the European Union. The new thing is that we must be able to indicate

– The rule concerns sales.
– The rule concerns goods.
– The place of delivery must be in the European Union.
– The buyer must be registered for VAT.
– You must acquire the VAT registration number of the company.

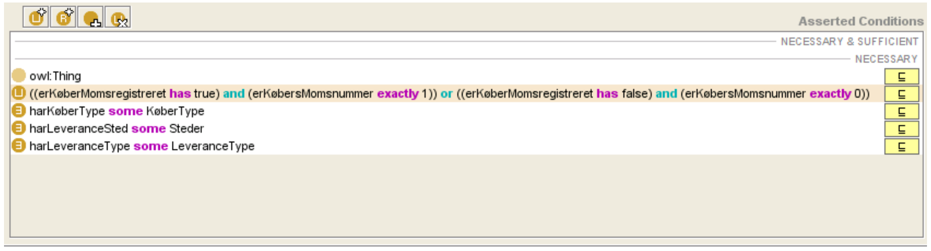
Table 6. Necessary & Sufficient conditions for application of the rule in Table 5.

whether a buyer is registered for VAT and if so, we must register the buyers VAT registration number. We use a functional data type property named `erKøberMomsregistreret` (translates into `isTheBuyerRegisteredForVAT`) with the data type `xsd:boolean` as its range to model whether the buyer is registered for VAT. Similarly we use a functional data type property named `erKøbersMomsnummer` (translates into `isBuyersVATRegistrationNumber`) with the data type `xsd:string` as its range to register the buyers VAT registration number if he has one.

A read through of [4] will reveal that you must register the VAT registration number of the buyer exactly when the buyer is registered for VAT. Thus we model this as a property of `VarerOgYdelser` and not of `Varer` (as indicated by the rule). The requirement can be modeled as follows:

– `((erKøberMomsregistreret has true)` $\sqcap$ `(erKøbersMomsnummer exactly 1))` $\sqcup$
`((erKøberMomsregistreret has false)` $\sqcap$ `(erKøbersMomsnummer exactly 0))`

The result is shown in Figure 7.



```
owl:Thing                                                                                                        ⊑
((erKøberMomsregistreret has true) and (erKøbersMomsnummer exactly 1)) or ((erKøberMomsregistreret has false) and (erKøbersMomsnummer exactly 0))   ⊑
harKøberType some KøberType                                                                                      ⊑
harLeveranceSted some Steder                                                                                     ⊑
harLeveranceType some LeveranceType                                                                              ⊑
```

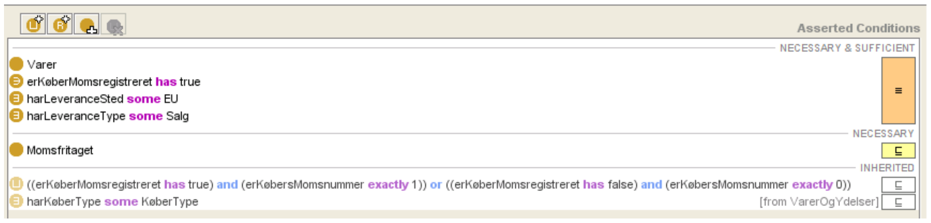
Fig. 7. Asserted Conditions of `VarerOgYdelser` after adding the requirement for registering VAT registration numbers.

5.2 Rule Modeling - Step II

Having added the necessary attributes to the model we are ready to model the rule itself. Since the rule describes a situation where you do not have to pay VAT we model it as a subclass of `Momsfritaget`. Following our naming convention we name the class `MomsfritagetSalgTilAndreEU-lande` (translates into `VATFreeSalesToOtherEUCountries`). Based on the necessary and sufficient conditions captured in Table 6 we add the following necessary and sufficient conditions to `MomsfritagetSalgTilAndreEU-lande`:

– `harLeveranceType some Salg`
– `Varer`
– `harLeveranceSted some EU`
– `erKøberMomsregistreret has true`

We note that the obligation to register the buyers VAT registration number is modeled indirectly, see Section 5.1. The result is shown in Figure 8.



```
Asserted Conditions
                                                  NECESSARY & SUFFICIENT
Varer                                                                  ≡
erKøberMomsregistreret has true
harLeveranceSted some EU
harLeveranceType some Salg
                                                               NECESSARY
Momsfritaget                                                           ⊑
                                                               INHERITED
((erKøberMomsregistreret has true) and (erKøbersMomsnummer exactly 1)) or ((erKøberMomsregistreret has false) and (erKøbersMomsnummer exactly 0))   ⊑
harKøberType some KøberType                          [from VarerOgYdelser] ⊑
```

Fig. 8. Asserted Conditions of our model of the legal rule in Table 5.

6 Future Work

The work presented in this document is work in progress and there are a lot of areas we need to address. In the near future we plan to integrate our model in a prototype ERP system as described in the introduction. This opens the possibility for modeling the parts of the Danish VAT legislation concerning depreciation and VAT reporting (since they are intertwined and contain a lot of technical requirements on the financial reports). We also need to model other countries VAT rules in order to confirm that Danish VAT rules are indeed representative with respect to the constructions that are needed in the modeling language. Based on this we need to refine our overall framework such that it captures the common structure and we need to identify what kinds of questions a model must be able to answer. The synthesized knowledge from modeling the VAT rules of other countries should also result in a more detailed analysis of what we can and cannot model.

Based on all this we should design a minimal description logic extended with the needed functionality identified in the analysis just mentioned, such as predicates like $x < 100$, which are needed in some rules. We should also provide a reasoner for the logic together with an editor such that the above process can be repeated.

Finally in order to compare our OWL model with a different approach we want to make a model using Datalog, which is the de facto standard language used to express rules in deductive databases, of the rules we have formalized in OWL already. It would also be interesting to try a hybrid solution (e.g., OWL plus a rule language like SWRL). This work is independent of the tasks mentioned above and can be carried out in parallel.

7 Conclusions

We have shown how to model a subset of Danish VAT rules concerning exemption from VAT using Protégé-OWL. First we created an overall framework for the VAT model with the property that legal rules and the concepts they involve can be modeled as subclasses of classes in the framework. This helps to ensure that related concepts are modeled in the same way and that a single concept is not modeled twice. The second step was an iterative process consisting of two steps repeated for each rule. The first step is to extend the model such that the rule in question can be modeled. This is done by modeling concepts from the legal source as classes in the model and by adding attributes to the necessary conditions of such classes. The second step is to model the rule itself. This is done by adding specific requirements for application of the rule to the necessary and sufficient conditions of the class modeling the rule.

The step by step iterative modeling has been working fine in practice and an extension to cover several different VAT and duty rates does not seem to be problematic as long as they do not require us to model restrictions such as $x < 100$, which is not supported directly in OWL. Apart from modeling inequalities we have not had problems with respect to modeling. One problem though is that reasoning about individuals in OWL models is not supported very well. Therefore we have tried to avoid the use of individuals wherever possible.

References

1. Ian Horrocks and Peter F. Patel-Schneider. Reducing owl entailment to description logic satisfiability. *Lecture Notes in Computer Science*, 2870/2003:17–29, September 2003.
2. A. Rector R. Stevens M. Horridge, H. Knublauch and C. Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. 2004.
3. F.P. Coenen T.J.M. Bench-Capon. Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, 1(1):65–86, March 1992.
4. ToldSkat. *Moms - fakturering, regnskab mv. (Vejledning E nr. 27) Version 5.2*, Jan 2005.

A System of VAT Rules

Morten Ib Nielsen

Department of Computer Science
University of Copenhagen, Denmark
mortenib@diku.dk

Abstract. This document presents our first attempt at creating a small but representative collection of VAT rules relevant in an ERP context. Here relevant means that the rules (ought to) influence the way ERP systems handle VAT and representative means that a formalism able to model the rules ought to be able to model any relevant VAT rule.
The document is intended to lay the foundations for benchmarking and systematic comparison of (the expressiveness of) different approaches to VAT modeling.
The document is *work in progress* and is currently based on English and Danish VAT rules from the legal guides [1], [2] and [3].

1 A System of VAT Rules

Below we present a small collection of rules we have found to be representative with respect to expressiveness for VAT rules relevant in an ERP context. The set of rules is inspired by and heavily based on English and Danish VAT rules from [1], [2] and [3].

We have split our presentation of the rules into two parts: *Definitions* and *Rules*.

Definitions

Supplies: Supplies can be of either goods or services.

Rates: Three VAT rates exist *standard* (17,5%), *reduced* (5%) and *zero-rate* (0%). Further goods and services can be exempt from VAT. The difference between being exempt from VAT and zero rate is that in the former case you cannot recover VAT on your own expenditures except in certain cases described in....

Reduced rate: The following are reduced rate supplies: domestic fuel or power, installation of energy-saving materials, women's sanitary products, children's car seats, residential conversions; renovations and alterations.

Zero rate: The following are zero rate supplies: books, newspapers, food bought in stores (not in restaurants, cafes or take-away food and drink), young children's clothing and shoes, exported goods, most prescriptions dispensed to a patient by a registered pharmacist and most public transportation services.

Exempt supplies: The following are exempt supplies: insurance, betting; gambling and lotteries, fund raising events by charities, education and training, services of doctors; dentists and undertakers, charges made to you by the Post Office.

Taxable supplies: Taxable supplies are defined as any supply[1] that is not exempt from VAT.

Ambiguous rate: If more than one VAT rate apply any of the rates in question can be applied.

Distance selling: Distance selling is when an entity in one European Union (EU) Member State supplies and delivers goods to a customer in another EU Member State and the customer is not registered for VAT, or liable to be registered for VAT. These customers are known as non-taxable persons.

Taxable person: A taxable person is an individual, firm, company etc who is, or is *required to be, registered for VAT*. You are required to be registered if the value of your taxable supplies exceed 64,000LCY[2] or your distance selling exceed 70,000LCY per year (1st January to 31st December).

Rules

1. No VAT (or refunds) for amounts less than 1 pound.
2. For VAT purposes, amounts of money must always be expressed in LCY. Rules regarding what exchange rate to use exist.
3. The appropriate rate for any supply (should there be changes) is that current at the time of the supply.
4. Normal and necessary packaging, including ordinary tins, bottles and jars, is treated as part of the goods which it contains. The price which your customer pays is treated as a payment for the contents of the packaging alone. This means that if your supply of the contents is zero-rated, then zero-rating also applies to the packaging.
5. A gift of goods is normally a taxable supply and VAT is due on the cost of the goods (see paragraph 7.6).
6. A gift of services is not a taxable supply.
7. If you make mixed supplies and the individual supplies are not liable to VAT at the same rate, you must work out the tax on each supply. If the tax value is based on the total price you must split the price between the supplies (called apportionment). Apportionment must be "fair" and justifiable but no complete rule set exists - guidance can be found in Notice 709/5.
8. Contingent discounts: If you offer a discount on condition that something happens later (for example, on condition that the customer buys more from you) then the tax value is based on the full amount paid. If the customer later earns the discount, the tax value is then reduced and you can adjust the amount of tax by issuing a credit note (see paragraph 18.2).
9. No VAT should be added on advertisers' novelties and samples you hand out free of charge if the purchase cost without VAT is less than 100 LCY.

[1] Most often of goods and services but other options exist.
[2] Local CurrencY unit.

10. No VAT should be added on necessary equipment and provisions delivered directly for use on ships and planes in international service.
11. No VAT should be added to goods delivered to destinations outside the European Union. This fact ordinarily also applies to services, but VAT should be added to certain services.
12. VAT should not be added to goods and transport services delivered to embassies and international organizations in countries within the European Union.
13. No VAT should be added to goods delivered to companies in other EU countries, provided that the companies are registered for VAT. In this case you must acquire the VAT registration number of the company.

References

1. HM Revenue & Customs. Notice 700: The VAT Guide – All Sections. HMRC, April 2002.
2. HM Revenue & Customs. Notice 700/1: Should I be registered for VAT? HMRC, October 2007.
3. ToldSkat. Moms - fakturering, regnskab mv., January 2005.

Modeling VAT Rules in CPML

Morten Ib Nielsen

Department of Computer Science
University of Copenhagen, Denmark
mortenib@diku.dk

Abstract. This document describes and evaluates our attempt to model the VAT rules contained in [3] using Configit PML and Product Modeller. In addition we describe CPML (the modeling language) and Product Modeller (the tool/IDE) briefly. We conclude that CPML is expressive enough to handle non-temporal VAT rules, but that it is not suitable for modeling of a full scale set of VAT rules due to lack of domain support.

1 Introduction

We begin with a short introduction to parts of the Configit Software Suite, a commercial system for building interactive sales and product configurators [2].

Product Modeller is a graphical tool for building (finite) *CPML models*, which consist of a set of variables over finite domains together with (logical) rules, relations and functions on them. Rules and relations specify interrelationships between variables in first order propositional logic, and are said to be applicable in "both directions" as opposed to functions, which specify a one-way relationship between variables.

It is possible to organize variables in a tree structured hierarchy using groups. The hierarchy has no semantic meaning when models are compiled (see below) but provides an easy way to control common parameters of several variables simultaneously (e.g., whether a set of variables is enabled or commented out). In addition groups are used to guide the layout of the *standard configurator*, see Figure 3.

The purpose of a CPML (Configit PML) model is to describe all legal assignments of values to variables plus possibly some derived information and to help (human) users to arrive at a legal combination (a configuration). Thus the problem we are faced with contains that of satisfiability, which is NP complete. Since it is easy to verify a solution to the configuration problem in polynomial time as well we conclude that the configuration problem itself is NP complete.

In order to give good runtime guarantees when product models are used, Configit has isolated the NP completeness of the problem in an offline compilation step where the so-called *VT-table* is generated, see Figure 1. The VT file, which essentially contains a binary decision diagram describing the full solution space, can be queried through the *Configit runtime* (e.g., by using the web based standard configurator shown in Figure 3).

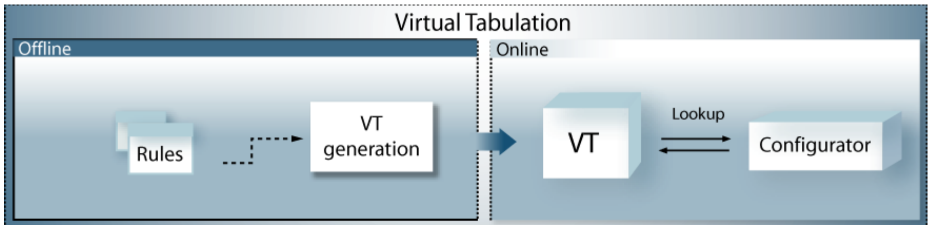


Fig. 1. Virtual Tabulation (source: [1]).

1.1 A Word on Variables

Variables in CPML models have a range of custom properties that control how you can interact with them through the standard configurator. The default is that variables are displayed and that you can change their value (if the constraints imposed by rules and relations allow you to) via the standard configurator. This behaviour is achieved by setting the properties `Show` and `ReadOnly` to true and false respectively, but the choice does not need to be static. In fact the runtime will reevaluate these properties whenever the configurator changes state, which enables you to dynamically control what variables a user can see and whether he can edit them.

Two other important custom properties are `ShowAs` and `CalculatedValue`. `ShowAs` controls how a variable is displayed in the standard configurator. Apart from options such as drop down and radio button rendering one can choose the special option `Calculated`. When this is done the standard configurator displays (if `Show` evaluates to true) the result of running the function specified in the `CalculatedValue` property. Calculated values are always read only and are often used to display derived information. Below we use a calculated value to display the `VATpercentage`.

2 Modeling VAT Rules in CPML

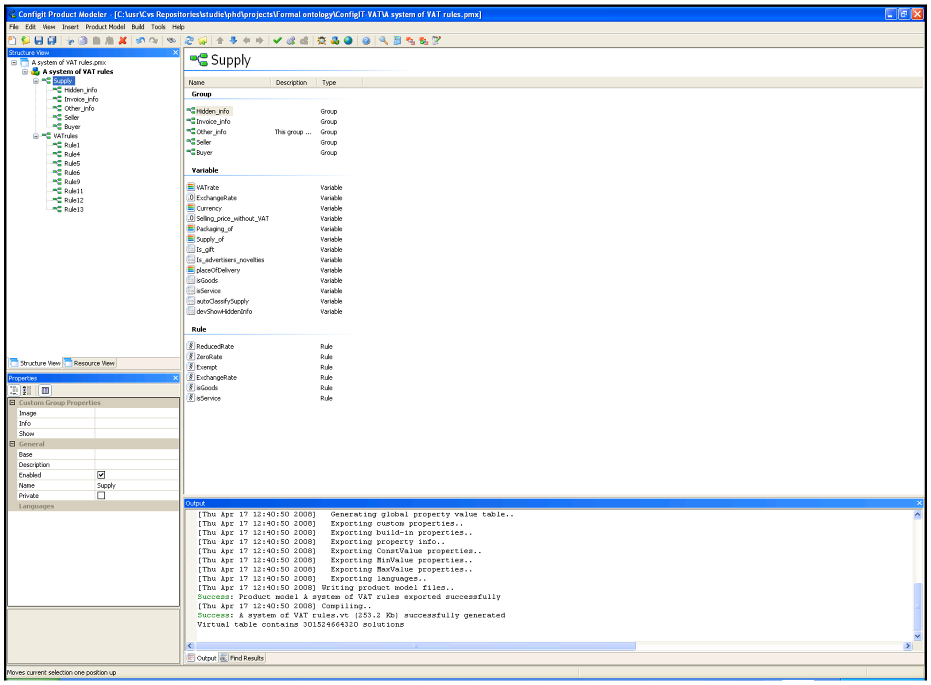
We have used Product Modeller to create a CPML model[1] of the VAT rules in [3]. The model covers all definitions except **Taxable person**, which was skipped because we assume companies running ERP systems are required to be registered for VAT. Furthermore we decided to focus on non-temporal rules only and thus rules 3 and 8 are excluded from the model. Finally we have not modeled rule 7 because it considers several lines[2], which is not supported by our model since it works in a line-by-line fashion similar to Microsoft Dynamics NAV.

The purpose of our model is to calculate the selling price without VAT, the VAT component and the VAT percentage of a given supply. The reason why not

[1] Mail `mortenib@diku.dk` to obtain a copy.
[2] A line corresponds to a quantity of a particular supply. Thus a quote or invoice covering multiple supplies must have several lines.

only one of the VAT component and the VAT percentage suffices is that the VAT component might be zero even though the selling price without VAT and the VAT percentage are not. An example is when rule 1 applies. In that case the VAT percentage might be different from zero depending on parameters such as type of supply, but the VAT component is zero because of the rule.

On the top-level our model consists of two groups `Supply` and `VATrules`. `Supply` contains variables (and sub-groups) describing a supply and rules modeling the definitions from [3]. For instance `Supply` contains variables `Supply_of` and `Is_gift` and rules `ZeroRate` and `Exempt` corresponding to definitions **Zero Rate** and **Exempt supplies**. `VATrules` contains a sub-group for each of the rules (except 3, 7 and 8) in [3]. A screen shot from the Product Modeller is provided in Figure 2.



Supply

Name	Description	Type
Group		
Hidden_info		Group
Invoice_info		Group
Other_info	This group ...	Group
Seller		Group
Buyer		Group
Variable		
VATrate		Variable
ExchangeRate		Variable
Currency		Variable
Selling_price_without_VAT		Variable
Packaging_of		Variable
Supply_of		Variable
Is_gift		Variable
Is_advertisers_novelties		Variable
placeOfDelivery		Variable
isGoods		Variable
isService		Variable
autoClassifySupply		Variable
devShowHiddenInfo		Variable
Rule		
ReducedRate		Rule
ZeroRate		Rule
Exempt		Rule
ExchangeRate		Rule
isGoods		Rule
isService		Rule

Custom Group Properties	
Image	
Info	
Show	
General	
Base	
Description	
Enabled	✓
Name	Supply
Private	☐
Languages	

```
[Thu Apr 17 12:40:50 2008]   Generating global property value table..
[Thu Apr 17 12:40:50 2008]   Exporting custom properties..
[Thu Apr 17 12:40:50 2008]   Exporting build-in properties..
[Thu Apr 17 12:40:50 2008]   Exporting property info..
[Thu Apr 17 12:40:50 2008]   Exporting ConstValue properties..
[Thu Apr 17 12:40:50 2008]   Exporting MinValue properties..
[Thu Apr 17 12:40:50 2008]   Exporting MaxValue properties..
[Thu Apr 17 12:40:50 2008]   Exporting languages..
[Thu Apr 17 12:40:50 2008] Writing product model files..
Success: Product model A system of VAT rules exported successfully
[Thu Apr 17 12:40:50 2008] Compiling..
Success: A system of VAT rules.vt (253.2 Kb) successfully generated
Virtual table contains 301524664320 solutions
```

Fig. 2. Screen shot from the Product Modeller. On the right hand side we see variables etc. in the group `Supply`.

We have tried to model the rules as close to their wording as possible. This means that we have taken care not to add knowledge that is not explicitly contained in the rules even though it might be the intention of the authors of the rules.

2.1 Modeling Precedence

Some of the rules (1, 9-13) describe situations where no VAT is due even though other rules say so. We say that these rules take precedence, i.e. they are more important. Since CPML does not support precedence on rules we had to model this and we did so in the following way: Each rule has a boolean variable called `LHS`, which is assigned the value `true` if and only if the rule applies. Furthermore each rule has a function called `RHS`, which in the case of rules 1 and 9-13 returns 0.0. The variable `LHS` and the function `RHS` are used in a function, `precedenceVATpayableInLCY`, containing the precedence hierarchy for calculation of the VAT component (we display the result as a calculated value of the variable `VAT_payable_in_LCY`). The function `precedenceVATpayableInLCY` contains one big `if-then-else if...` expression, where the first if-branch contains the rule with the highest precedence, while the last else-branch contains the default rule, which is to calculate the VAT component from the selling price and the VAT percentage.

An alternative way to specify the precedence hierarchy as a rule instead of a function is the following:

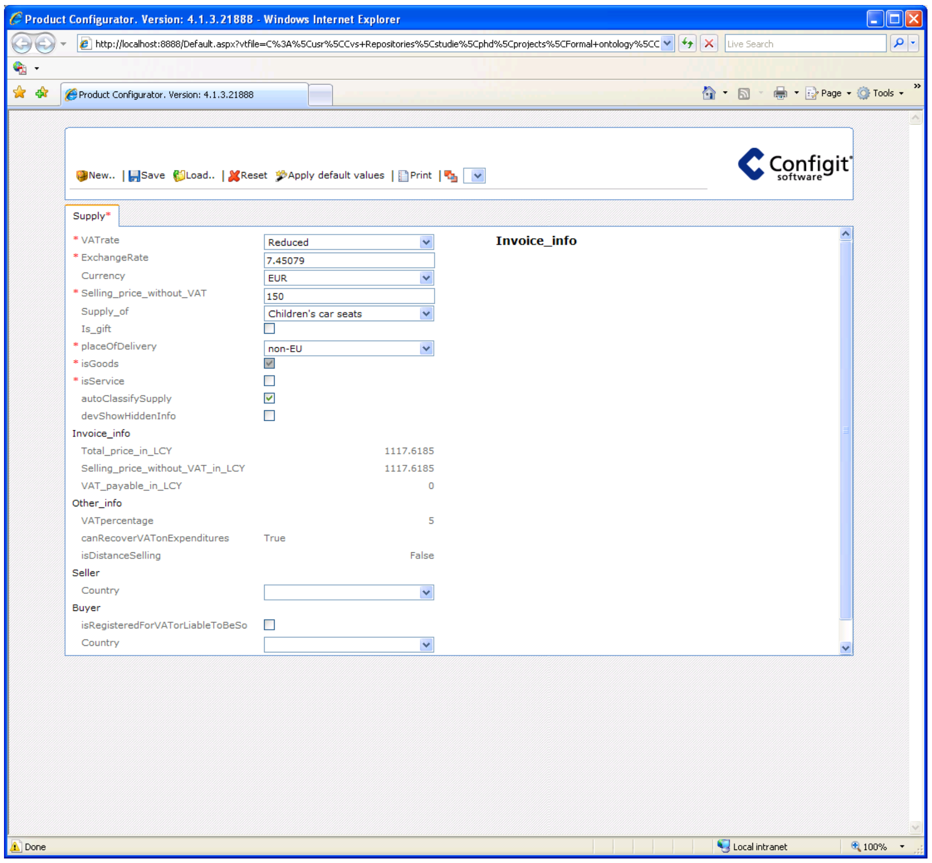
$$\begin{aligned}(LHS_1 \Rightarrow VAT\_payable\_in\_LCY = RHS_1) \wedge \\ (\neg LHS_1 \wedge LHS_2 \Rightarrow VAT\_payable\_in\_LCY = RHS_2) \wedge \\ \cdots \wedge \\ (\neg LHS_1 \wedge \neg LHS_2 \wedge \cdots \wedge LHS_n \Rightarrow VAT\_payable\_in\_LCY = RHS_n) \wedge \\ (\neg LHS_1 \wedge \neg LHS_2 \wedge \cdots \wedge \neg LHS_n \Rightarrow VAT\_payable\_in\_LCY = \texttt{<default calc>})\end{aligned}$$

The reason we have chosen to use a function, which is one-way, instead of a rule lies in `<default calc>`. Here we need to multiply two floating point numbers and this is not allowed in rules because CPML treats floats (and integers) as finite sets of intervals, which would become too large if multiplication was allowed.

As an illustration of precedence in action Figure 3 shows a screen shot from the standard configurator where the VAT rate for *Children's car seats* has been inferred to be *Reduced*, which means that the VAT percentage is 5. Never the less we see that no VAT is due (`VAT_payable_in_LCY` is zero). This is due to rule 11, which states that *No VAT should be added to goods delivered to destinations outside the European Union....*

2.2 Peculiarities

Since our model is intended only as a *proof of concept* we have cut a few corners along the way. For instance variables `Supply_of` and `Packaging_of` share the same (enum) type even though `Packaging_of` should not allow the option `Necessary packaging`. Another issue is exported goods, which the model supports in two ways. First one can explicitly state that `Supply_of` is `Exported goods`, but it is also possible to infer that a supply (different from `Exported`


Product Configurator. Version: 4.1.3.21888 - Windows Internet Explorer
http://localhost:8888/Default.aspx?vtfile=C%3A%5Cusr%5CCvs+Repositories%5Cstudie%5Cphd%5Cprojects%5CFormal+ontology%5CC
Live Search
Product Configurator. Version: 4.1.3.21888
Page
Tools
New..
Save
Load..
Reset
Apply default values
Print
Configit software
Supply*
Done
Local intranet
100%

Invoice_info

* VATrate	Reduced
* ExchangeRate	7.45079
Currency	EUR
* Selling_price_without_VAT	150
Supply_of	Children's car seats
Is_gift	☐
* placeOfDelivery	non-EU
* isGoods	✓
* isService	☐
autoClassifySupply	✓
devShowHiddenInfo	☐
Invoice_info	
Total_price_in_LCY	1117.6185
Selling_price_without_VAT_in_LCY	1117.6185
VAT_payable_in_LCY	0
Other_info	
VATpercentage	5
canRecoverVATonExpenditures	True
isDistanceSelling	False
Seller	
Country	
Buyer	
isRegisteredForVATorLiableToBeSo	☐
Country	

Fig. 3. Screen shot from the standard configurator where rule 11 takes precedence.

`goods`) should be treated as exported goods using rule 11. The reason both options exist is that the the rules in [3] have both of them. Thus the rules underlying the model are ambiguous in this respect.

3 Evaluation of VAT Modeling Using Configit PML

In this section we evaluate our attempt to model VAT rules in CPML.

3.1 Ease of use

Product Modeller is relatively easy to work with and can in the case of rules and relations be operated by people who do not have prior experience with computer science or modeling in particular. This fact ceases to be true, when we take functions into account since they must be written in a C-like programming language. Since CPML is a general purpose modeling language it lacks domain support for the legal domain such as precedence on rules, lex specialis, main rules with exceptions etc. These shortcomings complicate development of legal models.

3.2 Expressiveness

CPML contains a rule language with the expressiveness of first order propositional logic with equality over variables with finite domains plus ordering of floats and integers, which are modeled as finite sets of intervals. A consequence is that arithmetic expressions on integers and floats are not allowed in rules. Apart from the rule language CPML contains a Turing complete functional language. A significant difference between rules and functions is that functions are one way. A function specifies how to compute a set of output values given some set of input values. As a consequence hereof functions should only be used to compute derived information. As an example let us consider the difference between the rule `if A then B` and the function `if A then B:=true`. The rule is equivalent to the logical expression $\neg B \Rightarrow \neg A$, but the function is not.

Using rules and functions we were able to model the rules described in Section 2.

3.3 Reasoning

It is possible to reason about a CPML model if it can be compiled (remember compilation is NP complete). In our work we have not had problems in doing so and we do not know of any companies who have not been able to compile their (significantly larger) product models even though it sometimes takes some tuning of the (BDD) variable ordering.

3.4 Conclusion

We have found that CPML is expressive enough to handle modeling of non-temporal VAT rules, but also that CPML and Product Modeller do not support modeling in the legal domain (CPML is a general purpose modeling language). This together with the fact that functions must be written in a C-like programming language means that Product Modeller/CPML is not suited for use by domain experts in the legal domain. In conclusion we find that Product Modeller/CPML is not suited for creating a full sized model of (Danish) VAT rules. But since CPML is expressive and reasoning is fully supported we envision that a domain specific language tailored for the legal domain can be compiled to an equivalent CPML model, which can be used as the basis for a VAT query system.

References

1. Configit Software. The competitive edge of virtual tabulation (white paper). `http://www.configit.com/docs/exec_vt.pdf` (date of citation: 2008-06-06).
2. Configit Software. Configit product modeler, version 4.1. Limited edition demo available from `http://www.configit.com`.
3. Morten Ib Nielsen. A System of VAT Rules. Mail mortenib@diku.dk to obtain a copy, 2007.

Logical Models for Value-Added Tax Legislation

Morten Ib Nielsen[*] Jakob Grue Simonsen
Ken Friis Larsen[*]
Department of Computer Science, University of Copenhagen, Denmark
{mortenib|simonsen|kflarsen}@diku.dk

Abstract

Enterprise resource planning (ERP) systems are ubiquitous in commercial enterprises of all sizes and invariably need to account for the notion of value-added tax (VAT). The legal and technical difficulties in handling VAT are exacerbated by spanning a broad and chaotic spectrum of intricate country-specific needs. Currently, these difficulties are handled in most major ERP systems by customising and localising the native code of the ERP systems for each specific country and industry. We propose an alternative that uses logical modeling of VAT legislation. The potential benefit is to eventually transform such a model automatically into programs that essentially will replace customisation and localisation by configuration by changing parameters in the model. In particular, we: (1) identify a number of requirements for such modeling, including requirements for the underlying logic; (2) model salient parts of the Danish VAT law in Web Ontology Language (OWL) and in Configit Product Modeling Language (CPML). We conclude that OWL is ill-suited for VAT modeling due to the lack of support for integer inequalities and certain constructions on data type properties, while CPML, its excellent tool support notwithstanding, does not offer the right (high) level of abstraction to be practical for modeling VAT.

1 Introduction

Virtually all businesses employ an *enterprise resource planning* (ERP) system[1] to help them keep track of the economical aspects of their business. Most of these ERP systems are designed and programmed in a generic fashion and later heavily adapted to particular geographical regions and specialised industries, indeed often heavily adapted to the specific company at which the system is deployed.

The adaptations to an ERP systems can be divided into two kinds: *customisations*, requiring changes to the source code of the systems, and *configurations*, requiring changes to be interpreted in an overlay system at runtime.

We call the adaptations made to an ERP system when it is to be deployed in a specific country *localisation*. Localisation is both the translation of the user-visible strings in system and, more importantly, the changes to the system with respect to the local legislation. Localisation can be performed both as customisation and as configuration.

One area of paramount importance for a given company is value added tax (VAT) and all of today's standard ERP systems handle VAT, typically in a semi-automated way requiring a large degree of user interaction. While VAT rules from a birds-eye view are seemingly simple there are lots of corner-cases, especially in international trade, that we cannot expect an ordinary user of an ERP system to know of.

Currently ERP vendors handle VAT localisation by the following idealised process: First, a domain-expert in VAT legislation translate the relevant parts of the VAT law into a natural-language specification. Second, a programmer implements in code the specification written by the domain-expert. In practise, this specification is passed through a series of domain-experts of up to several dozens of people before the specification reaches a team of programmers.

We suggest to make all VAT localisation configuration, and to give this configuration in the form of a explicit logical *VAT model*. The benefits of this approach is to (1) drastically cut down the number of domain experts involved in the design process, (2) alleviate the implementation tasks of the programmer

[*]Supported by the Danish National Advanced Technology Foundation through the 3gERP project.

[1]We use the term ERP system in a broad sense in this article.

(as sizable chunks of the necessary code can be auto-generated), and (3) move the finished product closer to a state where it can be model-checked against a formalisation of the VAT legislation.

In this paper, we take the initial steps towards this goal by identifying a number of requirements needed by logic-based tools for VAT modeling and by detailing our experiments with two such tools: OWL and CPML.

2 Modeling of VAT Legislation

Legal rules are usually accessible through legal documents such as the EU directive on the common system of value added tax, [1], or through official guidelines (that also carry legal weight) such as [2] and [3]. While the (legal) semantics of legal documents and guidelines are (at least intended to be) the same, the latter are usually easier to understand for laymen but carry less (formal) structure than legal documents. We use the term *legal sources* to refer to both legal documents as well as official guidelines.

Structure in legal documents plays an important role for instance in connection with references between exceptions and main rules, which are often separated from each other (and might even occur in different documents). Another use of structure is in connection with meta principles referred to as *rules of collision* in legal doctrine. The purpose of these principles is to disambiguate in situations where legal rules are in conflict with each other. In this paper we only focus on *Lex Specialis* (particular norms suppress general norms).

3 Desired properties of a logical model

This section accounts for some of the desired properties of logical models of legal rules. We use F to denote a formal language and F-model to mean a model of legal rules written in F.

Good Tool Support We envision that F-models are created in part by domain experts without a strong background in computer science or modeling. Thus modeling in F should be easy and provide good domain support. It is not reasonable to expect that domain experts interact with F directly, therefore we envision a GUI as a mediator between domain experts and F-models. Even though (usability) design of a GUI is outside the scope of this paper the interplay between GUI and F is important. One way to design a GUI is by using graphical notation as sugar for common constructions in F much like it is done in [4].

Support the isomorphism principle This principle, as presented in [5], states that:

> A representation of a legal system is an isomorphism if (1) Each legal source is represented separately. (2) The representation preserves the structure of each legal source. (3) The representation preserves the traditional mutual relations, references and connections between the legal sources. (4) The representation of the legal sources and their mutual relations is separate from all other parts of the model, notably representation of queries and facts management.

For our application support for structure and references, e.g. loose coupling of main rule and exceptions seems to be the most difficult to achieve.

Conform to legal doctrine As described in Section 2, it is important to be able to express rules with exceptions and the support for meta principles such as Lex Specialis.

Be transferable Modeling of legal sources will most likely involve many people with different legal specialisations. Therefore, the combination of F and its accompanying editor should support good software engineering principles, including support for transfer of the model between persons with different areas of responsibility.

Sufficient expressive power F must have the ability to express the semantics of relevant rules and properties of the problem domain. It is clear that F must be able to handle at least Presburger arithmetic (amongst other things due to the need for multiplication with VAT percentage constants) on integers. Good domain support also requires the use of variables over general finite domains, which is the reason why we have not considered using a SAT solver (even though any finite domain can be coded).

Amenable to static analysis It is desirable that F supports decidable reasoning, e.g. static analysis such as dead rule detection and inconsistency checks.

Have tractable reasoning Another crucial feature is the existence of a well-maintained reasoner with the ability to handle large models (the EU Directive on VAT is 118 pages) and simultaneously process many transactions (queries) at runtime—in short reasoning must be tractable. We note that expressive power and tractability of reasoning are opposite forces.

We believe that the the need for expressive power is best analysed by evaluating existing formalisms and tools. We report on such an evaluation in the next section.

4 Modeling in Two Logic-based Tools

We now report on our experiences using OWL (description logic) [6] and Configit Product Modeling Language (CPML) [7] to model select legal rules. We have chosen OWL and CPML because they seemed prime candidates due to their (on the surface) satisfying most of the requirements of the last section, especially on the side of tractable reasoning and tool support. In case of OWL we have used the Protégé editor [8] and the Racer Pro reasoner [9] while CPML comes with its own complete tool suite.

The litmus test is the ability of the tools to handle a suite of extremely basic and common tasks associated with VAT handling in ERP systems: To calculate the selling price without VAT, the VAT component and the VAT percentage of a given supply in a line-by-line fashion similar to Microsoft Dynamics NAV. A line corresponds to a quantity of one particular supply. Thus if a quote or invoice contains multiple supplies it must have several lines. The reason why not only one of the VAT component and the VAT percentage suffices is that the VAT component might be zero even though the selling price without VAT and the VAT percentage are not. An example is rule 4 in Figure 1.

We have evaluated OWL and CPML against a list of legal definitions and rules which we believe are representative for the legal domain. In order to give the reader a hint about the flavour of legal rules we present a few in Figure 1.

4.1 OWL

OWL, short for Web Ontology Language, is an ontology language designed to be compatible with the World Wide Web and the Semantic Web[6, 10, 11]. The most salient abstraction in OWL is a concept axiom, also called a *class*. Each class has a list of necessary conditions and zero or more equivalent lists of necessary and sufficient conditions governing membership of the class. OWL comes in three variants: OWL-Lite $\subseteq$ OWL-DL $\subseteq$ OWL-Full of increasing expressive power. The variants OWL-Lite and

1. *Samples and items used for advertising. No VAT should be added to items used for advertising or the like if the value (purchase price) is less than 100 kr. exclusive of VAT nor to samples you hand out for free.* Translated from [3, p. 8].
2. *Sales outside EU (3rd countries). No VAT should be added to goods delivered to destinations outside the European Union, or to the Faroe Islands or Greenland. This fact ordinarily also applies to services, but VAT should be added to certain services.* Translated from [3, p. 9].
3. *Sales in other EU countries. No VAT should be added to goods delivered to companies in other EU countries, provided that the companies are registered for VAT. In this case you must acquire the VAT registration number of the company.* Translated from [3, p. 8].
4. *If the net VAT payable calculated to the nearest penny, is less than £1, no payment need be made.* From [2, p. 100].

Figure 1: Legal rules.

OWL-DL are based on the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ respectively, which guarantees the decidability of important inference problems such as satisfiability and subsumption.

Structure of the model: We created our OWL model in an iterative way starting from a small framework of classes central to VAT such as `Location`, `Goods`, `Services` and `FreeOfVAT` together with properties such as `hasDeliveryType` and `hasDeliveryPlace`. We have used value partitions (design pattern, [10][p. 73-76]) to model attributes with finite domains and data type properties for infinite domains. Modeling of the rules then progressed by first extending the model with needed classes, data type properties etc. and second adding a class with necessary and sufficient conditions corresponding to the rule in question. Sometimes we needed to add extra info in other places, for an example see Figure 3 part 3.

Evaluation and discussion: The primitives in OWL and the tool support we had did not fit well with modeling of legal rules. We often found that the addition of a new rule would trigger a global change in the model due to ill-preservation of the structure in the legal source. There is one area, however, where OWL did provide good support namely the in the modeling of definitions and their interrelationship, which we could term *ontology of the legal domain*. A salient problem with OWL was the lack of support for inequalities on integers which prevented us from modeling rules 1 and 4 in Figure 1. This taken together with the (state of the art) reasoners ill support for individuals, which we have avoided to use for this reason, and lacking support for hasValue ($\ni$) and quantifier restrictions ($\exists$) on datatype properties (e.g., `isBuyerRegisteredForVAT has true` and $\exists$ `isBuyerRegisteredForVAT some boolean`) makes us conclude that reasoning about the legal domain is not tractable using OWL. Due to the problems with expressivity we have not been able to experiment with analysis on OWL models (one of our ideas was to use the reasoner for dead rule detection (concept satisfiability) and inconsistency checks). Finally we have not been able to find any reasonable way to implement support for legal doctrine, e.g. Lex Specialis.

4.2 CPML

We call the language used in *Configit Product Modeller* (Product Modeller henceforth) for CPML[7]. Product Modeller is a graphical tool for building *CPML models*. A CPML model consist of variables with finite domains together with rules, relations and functions. Rules and relations specify dependencies

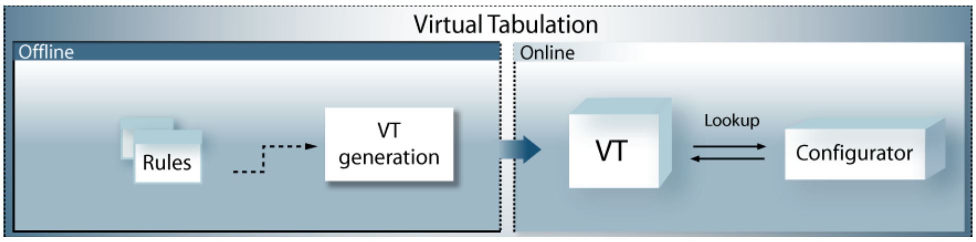


Figure 2: Virtual Tabulation (source: [12]).

between variables in first order propositional logic. Variables etc. can be organised in groups which form a tree structured hierarchy. The groups provide means to control common parameters easily, such as whether a group is currently enabled or commented out, and also guide the layout of the *standard configurator* (see Appendix C, Figure 5 and 6).

The purpose of a model is to describe all legal assignments of values to variables and to help a (human) user to arrive at a legal combination (a configuration). In general, this problem is NP-complete. In order to give good runtime guarantees Configit has isolated the NP-hardness of the problem in an offline compilation step, see figure 2. The output of the compilation is a *VT file*, short for Virtual Tabulation, containing a binary decision diagram describing the full solution space.

Structure of the model: On the top level our model consist of two groups: `Supply` containing variables (and sub-groups) describing a supply and `VATrules` containing a sub-group for each of the rules we have modeled. A screen shot from the Product Modeller is included in the appendix (Figure 5).

Modeling precedence: All rules in Figure 1 describe situations where the VAT component is zero even though other rules conclude differently. This is due to Lex Specialis, see Section 2. CPML does not support precedence so we need to model this. Our solution was to create a variable named *LHS* for each of the rules, which is true exactly when the corresponding rule applies, and to overwrite the VAT component with the content of a rule specific variable named *RHS*. The update was controlled using a simple `if` LHS_1 `then` ...`else if` LHS_2 `then` ... `else <default>` function.

One might think that we could have specified the precedence hierarchy as a rule as follows:

$$\big(LHS_1 \Rightarrow VAT_{payb} = RHS_1\big) \wedge \big(\neg LHS_1 \wedge LHS_2 \Rightarrow VAT_{payb} = RHS_2\big) \wedge \cdots \wedge \\ \big(\neg LHS_1 \wedge \neg LHS_2 \wedge \cdots \wedge \neg LHS_n \Rightarrow VAT_{payb} = \texttt{<default>}\big)$$

but this is not possible because `<default>` contains a multiplication which is not allowed in rules because Configit models numbers as finite sets of intervals (which would become too large if multiplication was allowed). An example where precedence kicks in is given in Figure 6 in the appendix.

Evaluation and discussion: CPML comes close to realising the isomorphism principle but cannot separate main rules from exceptions. Furthermore the Product Modeller is easy to work with but does not support the legal domain. An example is the limited support for precedence, which was only available through the use of functions. Finally we conclude that the expressive power of CPML is sufficient and that reasoning is tractable provided that the model compiles (which is NP complete but not a problem in practise).

5 Related work

Our work is related to the large body of work on formalisation of and automated reasoning about legal rules. An early example of applying logic programming to legal rules is the formalisation of the British

Nationality Act by Sergot et al. [13]. Such systems attempt (in principle) to replace domain experts. In [14] Leith argues that this is not feasible in the legal domain essentially because of what Prakken refers to as the *open texturedness of legal concepts*.

Open texture as presented by Prakken in [15] is the problem of interpretation of legal concepts including the mapping of real world objects to legal concepts. Prakken [15, p. 49-55] identifies four kinds of open texturedness: *underdetermination*; the situation where it is not clear whether an instance is a member of a concept, *overdetermination*; where conflicting rules exist, *defeasibility of legal concepts*; the situation where unspecified (unforeseen) exceptions might exist and *vagueness*; covering rules using terms such as reasonable, sufficient, and so on.

Prakken [15, p. 59] suggests the use of non-monotonic logics to handle problems of open texture as well as separation of main rule and exceptions. An immediate consequence is that all of the KB must be searched for every query. Since our goal is a more practical one, namely to provide better support for the complexity of VAT in ERP systems, we assume that domain experts resolve under- and overdetermination when they write the model, that the model is complete or able to give *don't know* answers and that vagueness is resolved through dialogue with the users.

6 Conclusion and future work

We have proposed using logical modeling of VAT legislation and assessed the benefits of this approach. We have identified a number of requirements for such modeling, including requirements for the underlying logic, and we have modeled salient parts of the Danish VAT law in two different logic-based tools: Web Ontology Language (OWL) and Configit Product Modeling Language (CPML). We found that: (1) OWL affords primitives that are well-suited for modeling legal definitions, but ill-suited for modeling legal rules, that OWL did not satisfy the crucial need for integer inequalities and that state-of-the-art tools for OWL do not support certain crucial logical operations. (2) CPML apparently satisfies all needs for modeling VAT legislation and promises tractable runtime reasoning due to the virtual tabulation compilation approach, but does carry some quirks, in particular it is not possible to separate the modeling of rules and their exceptions. In short: OWL does definitions well, CPML does rule modeling and execution well.

To obtain a good ontological fit to the domain of VAT modeling, a viable approach for future work will be to design a domain-specific language combining the excellent expressiveness of OWL with respect to legal definitions with the computational expressiveness and efficiency of CPML.

Furthermore, full modeling of the EU VAT directive and the VAT legislation of several large European nations must be performed.

Finally, logical modeling affords the possibility of doing provably correct analyses of a company's finances with respect to the VAT aspect of sales and purchases. This includes what-if scenarios, simulations and conformance checks, all of which are of potential economic benefit to a given company.

References

[1] Heinäluoma, E.: Council Directive 2006/112/EC of 28 November 2006 on the common system of value added tax. Official Journal of the European Union (English edition) **Volume 49 11**(L 347) (December 2006)

[2] Customs, H.R..: Notice 700: The VAT guide–all sections. HMRC (April 2002)

[3] ToldSkat: Moms - fakturering, regnskab mv. (January 2005)

[4] van der Aalst, W., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: The Role of Business Processes in Service Oriented Architectures. Number 06291 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany (2006) `http://drops.dagstuhl.de/opus/volltexte/2006/829`.

[5] T.J.M. Bench-Capon, F.C.: Isomorphism and legal knowledge based systems. Artificial Intelligence and Law **1**(1) (March 1992) 65–86

[6] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Owl web ontology language reference

[7] Configit Software Kristianiagade 7, DK-2100 Copenhagen Ø: Configit Product Modeler User's Guide. (June 2008)

[8] Haarslev, V., Möller, R., eds.: Proceedings of the 2004 International Workshop on Description Logics. Volume 104., CEUR-WS (2004)

[9] Sure, Y., Corcho, Ó., eds.: EON2003, Evaluation of Ontology-based Tools, Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools at ISWC 2003. In Sure, Y., Corcho, Ó., eds.: EON. Volume 87 of CEUR Workshop Proceedings., CEUR-WS (2003)

[10] M. Horridge, H. Knublauch, A.R.R.S., Wroe, C.: A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf (2004)

[11] Horrocks, I., Patel-Schneider, P.F.: Reducing owl entailment to description logic satisfiability. Lecture Notes in Computer Science **2870/2003** (September 2003) 17–29

[12] Configit Software: The competitive edge of virtual tabulation (white paper) `http://www.configit.com/docs/exec_vt.pdf` (date of citation: 2008-06-06).

[13] Sergot, M.J., Sadri, F., Kowalski, R.A., Kriwaczek, F., Hammond, P., Cory, H.T.: The british nationality act as a logic program. Commun. ACM **29**(5) (1986) 370–386

[14] Leith, P.: Fundamental errors in legal logic programming. Comput. J. **29**(6) (1986) 545–552

[15] Prakken, H.: Logical tools for modelling legal argument : a study of defeasible reasoning in law. Volume 32 of Law and Philosophy Library. Kluwer Academic Publishers (1997)

A Acknowledgements

The work presented in this article is carried out within the scope of the *3rd Generation Enterprise Resource Planning Strategic Software for Increased Globalisation* project partly funded by the Danish National Advanced Technology Foundation.

Also we would like to thank Configit Software who have provided us with an academic license to their software together with a training course.

B Example code

Models are available for download at `http://www.diku.dk/~mortenib/2008/lpar/models.zip`.

B.1 OWL

In Figure 3 we show the necessary and sufficient conditions used to implement rules 2 and 3, from Figure 1 in OWL. We cannot implement rules 1 and 4 in any reasonable way using OWL because inequalities on integers are not supported.

2.
```
 GoodsAndServices
∃ hasDeliveryPlace some non-EU
∃ hasDeliveryType some Sale
```

3.
```
 GoodsAndServices
isBuyerRegisteredForVAT has true
∃ hasDeliveryPlace some EU
∃ hasDeliveryType some Sale
```

Figure 3: OWL code (Protégé syntax) implementing rules 2 and 3 from Figure 1.

In case 3 we also add the following code to the necessary conditions on `GoodsAndServices`:

```
  ((isBuyerRegisteredForVAT has true) ⊓ (isBuyersVATRegNumber exactly 1))
⊔
((isBuyerRegedForVAT has false) ⊓ (isBuyersVATRegNumber exactly 0))
```

B.2 CPML

In Figure 4 we show the CPML code that implements the rules in Figure 1.

1.
```
 LHS <-> [Supply.Is_advertisers_novelties] and
[Supply.Invoice_info.Selling_price_without_VAT_in_LCY] < 100
```

2.
```
 LHS <-> ([Supply.isGoods] or [Supply.isService]) and
[Supply.placeOfDelivery] = "non-EU"
```

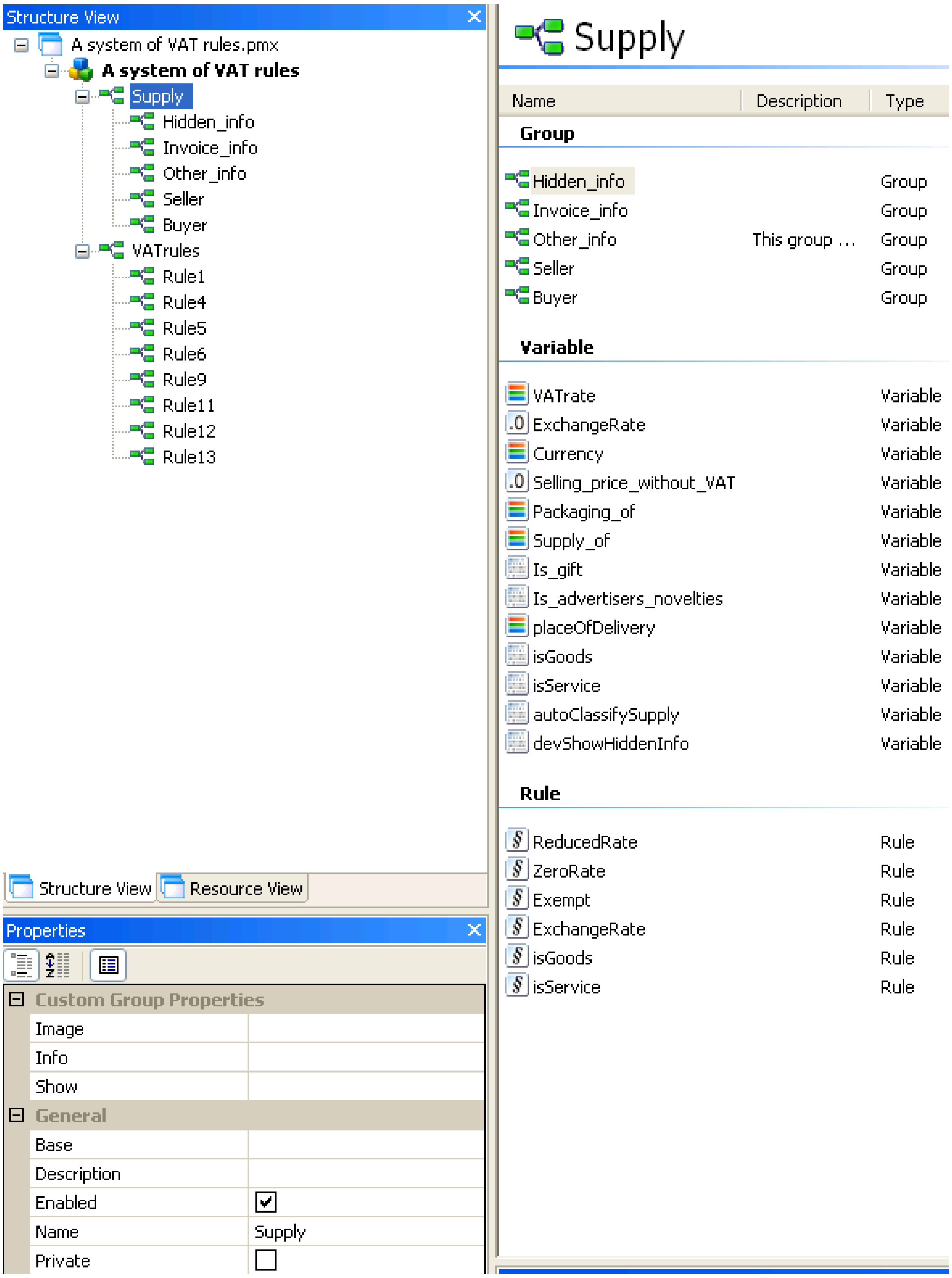
3.
```
 LHS <-> [Supply.isGoods] and [Supply.placeOfDelivery] = "EU" and
[Supply.Seller.Country] <> [Supply.Buyer.Country] and /* Models
"other EU countries" */ [Supply.Buyer.isRegisteredForVATorLiableToBeSo]
```

4.
```
 priceLessThan1LHS <-> [Supply.Invoice_info.Selling_price_without_VAT_in_LCY]
< 1
```

Figure 4: CPML code implementing the rules in Figure 1.

In case 3 we also use `LHS` to control whether the VAT number is required (a property on the VAT number variable).

C Figures



Structure View

- A system of VAT rules.pmx
  - **A system of VAT rules**
    - Supply
      - Hidden_info
      - Invoice_info
      - Other_info
      - Seller
      - Buyer
    - VATrules
      - Rule1
      - Rule4
      - Rule5
      - Rule6
      - Rule9
      - Rule11
      - Rule12
      - Rule13

Structure View | Resource View

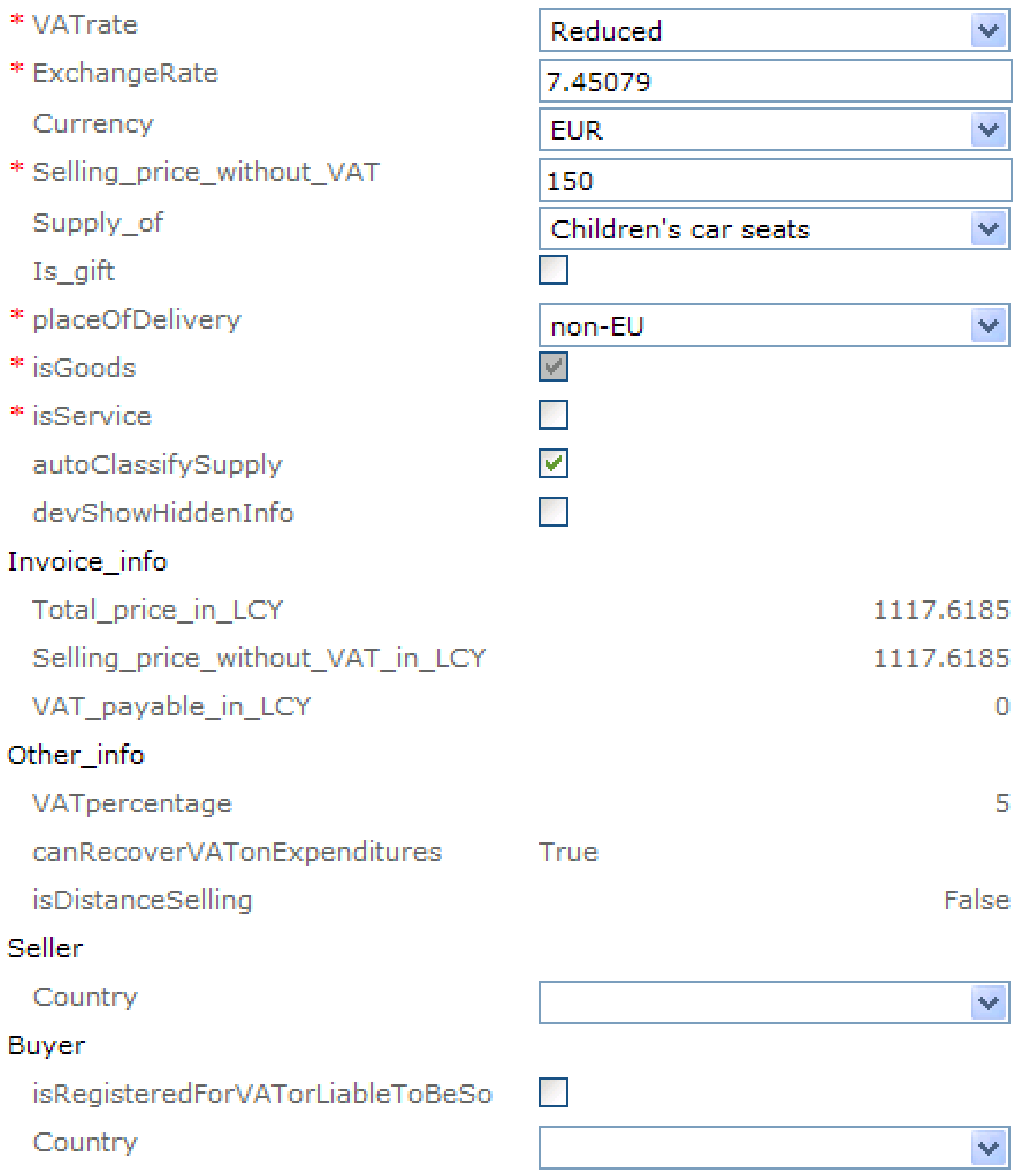
Properties

Custom Group Properties	
Image	
Info	
Show	
General	
Base	
Description	
Enabled	✓
Name	Supply
Private	☐

Supply

Name	Description	Type
Group		
Hidden_info		Group
Invoice_info		Group
Other_info	This group …	Group
Seller		Group
Buyer		Group
Variable		
VATrate		Variable
ExchangeRate		Variable
Currency		Variable
Selling_price_without_VAT		Variable
Packaging_of		Variable
Supply_of		Variable
Is_gift		Variable
Is_advertisers_novelties		Variable
placeOfDelivery		Variable
isGoods		Variable
isService		Variable
autoClassifySupply		Variable
devShowHiddenInfo		Variable
Rule		
ReducedRate		Rule
ZeroRate		Rule
Exempt		Rule
ExchangeRate		Rule
isGoods		Rule
isService		Rule

Figure 5: Screen shot from Product Modeller. On the right hand side we see variables etc. in the group `Supply`.



* VATrate	Reduced
* ExchangeRate	7.45079
Currency	EUR
* Selling_price_without_VAT	150
Supply_of	Children's car seats
Is_gift	☐
* placeOfDelivery	non-EU
* isGoods	✓
* isService	☐
autoClassifySupply	✓
devShowHiddenInfo	☐
Invoice_info	
Total_price_in_LCY	1117.6185
Selling_price_without_VAT_in_LCY	1117.6185
VAT_payable_in_LCY	0
Other_info	
VATpercentage	5
canRecoverVATonExpenditures	True
isDistanceSelling	False
Seller	
Country	
Buyer	
isRegisteredForVATorLiableToBeSo	☐
Country	

Figure 6: Precedence controls the VAT component. Here the VAT rate is Reduced (5%) but the VAT component is zero due to the second rule in Figure 1.