University of Helsinki
Department of Modern Languages
Language Technology

Master's thesis

Ontology-based lexicon management in a multilingual translation system — a survey of use cases

Inari Listenmaa
013302818

Supervisor: Lauri Carlson

November 9, 2012

Contents

1 Introduction

This thesis is a feasibility study of multilingual lexicon management with the help of ontologies. It is written as a part of MOLTO (Multilingual Online Translation), a translation technology project funded by the European Union. MOLTO provides a set of tools for creating small, domain-specific grammars that can be used for machine translation or collaborative multilingual authoring. MOLTO tools are targeted especially for content producers, such as administrators of multilingual websites. The idea is that the users of MOLTO tools can build a translation system for their needs, and the resulting system will be highly specialised to the domain, providing quality at the expense of coverage.

The tools are based on the interlingual paradigm of machine translation, such that all languages in the system have a two-way mapping to a common, language-independent representation of meaning; the technical details are described later, in Chapters 3 and 4. In addition to the tools, MOLTO will implement as case studies three complete translation systems, in the domains of mathematical exercises, museum object descriptions and pharmaceutical patents. This thesis will present concrete examples from the case studies to illustrate the problems and the solutions we have encountered.

In translating free text, ambiguity is a problem when the system is analysing the source text. In contrast, with domain-specific grammars, a more relevant problem is encountered when filling the gaps in lexicon. Let us take an example scenario. A user is writing a restaurant menu in English, which is being automatically translated into other languages supported by the multilingual grammar. The grammar has better support for some languages than others; for example, it has a wide selection of fish names in English, but not necessarily so in other languages.

Suppose that the user adds roasted bass to the menu, but there is no translation for bass in Finnish. The user will want to modify the lexicon on the fly, so that the translation process can continue. In the domain of the grammar, *bass* is not ambiguous: the grammar is about items that are in the restaurant menu, and it can only refer to a fish, not a low-voiced singer or an instrument. However, if the grammar lacks the word for this concept in some language, a user with no knowledge of Finnish would have to search for *bass*

in an English-Finnish dictionary. Dictionaries often do not provide sufficient context, and that is where the ambiguity problem arises.

Instead of dictionaries, we use *ontologies* as the source of the lexicon. An ontology is a hierarchy of concepts, where the terms are grouped by meaning, not just word forms. This helps to find terms that are related to the one that is searched. Suppose the user wants to add at the same time all things labeled as seafood; instead of searching just a word for bass and just in Finnish, she could do a search that gives all things similar to bass and get their translations in all languages the ontology covers.

The main purpose for ontologies in this thesis is using the concept structure as the source of lexicon, but we also touch the topic of using ontological reasoning for the disambiguation of natural language. Chapter 2 introduces the theoretical background, and Chapter 3 the tools and resources. Chapter 4 describes the use cases: disambiguation, lexicon harvesting and ontology verbalisation. Chapter 5 discusses related work, and Chapter 6 presents conclusions.

2 Theoretical background

This section covers the theoretical backround of this study: theories and methods related to machine translation and ontologies.

2.1 Machine translation

Machine translation is a branch of natural language processing (NLP). Its purpose is to automate translation from one natural language to another, either the whole process or parts of it (Arnold et al., 1993). It involves both natural language understanding and natural language generation, which makes it one of the hardest problems in computational linguistics. In the following subsections we will go through a short history of machine translation, introduce the machine translation paradigms and discuss the concept of ambiguity in machine translation.

2.1.1 History

This section presents a brief overview of the historical development of machine translation. For a more thorough survey, see e.g. Hutchins (2006).

The history of machine translation dates from the advent of the modern digital computer, that is, late 1940s to early 1950s. The problem was initially considered cryptographic instead of linguistic; the early approach was basically word-by-word direct translation by means of a bilingual dictionary and rules to fix the word order in the target language. By 1960s, there had been no significant breakthrough, and funding for machine translation was reduced in favour of more basic research in computational linguistics.

In 1970s and 1980s, machine translation became more linguistically ambitious. Basic research had created new methods for morphological and syntactic analysis. The MT systems were mostly based on human-written rules, ranging from mapping syntactic structures between two languages to trying to capture a completely language-independent meaning—the latter proved infeasible on a large scale, while the former produced some useful results. A famous example from this era is METEO, a rule-based system for translating weather reports between French and English (Slocum, 1984). However, the success stories of rule-based methods were mostly restricted to a specific domain, with a limited choice of vocabulary and a strict format.

By the 1990s, computational power had grown enough to enable new approaches, most importantly the statistical paradigm. Instead of human-written rules, statistical methods are based on learning from data; the linguistic representation does not depend on a priori categories, but empirical counts of what is attested in the corpus. This era has brought general-purpose MT systems for unrestricted text. Before the Internet was common, the main users of machine translation were companies, with the objective to replace at least parts of the work of professional translators. Although commercially developed machine translation aimed for publishers still exists, a typical example of modern MT is a system that is freely available for everyone and whose translation quality is poor but usable, intended for providing a rough idea of a text.

2.1.2 Machine translation paradigms

Some paradigms were mentioned briefly in the history of machine translation. In this section we get a more detailed view of the interlingual paradigm that is used in the MOLTO project, and for comparison, the statistical paradigm.

Interlingual machine translation An *interlingua* is a language-independent representation of meaning. The term interlingua often implies the use of such representation as an aid for translation, but its origin is in the ideas of 17th century philosophers, such as Gottfried Leibniz, René Descartes and John Wilkins, about a universal language that could express every human thought as a composition of simple concepts. For a long time, the idea remained only a philosophical one. Only in the 1970s, after more than two decades of machine translation research, it was put into use as a machine translation paradigm.

Interlingual machine translation is one of the rule-based paradigms, along with *transfer* and *direct translation*. What is common to all approaches is the application of linguistic information; some kind of mapping between the source and the target, in such a way that we know in each step the relation between each item. The three approaches can be placed on a scale depending on the level of intermediary representation from lowest to highest: direct translation, transfer-based translation and interlingual translation.

The simplest rule-based system is one that processes the source text one word at a time, (analyses each word morphologically,) searches the word in a bilingual dictionary, (inflects it accordingly in the target language) and outputs the translation. This requires a bilingual dictionary, either having all inflected forms as atomic units, or in combination with morphological analysis of both source and target languages. The intermediate representation is limited to morphology, and context limited to one word. This system can be improved somewhat, for example, with additional rules to modify the word order in the target language, but the defining feature is the dictionary mapping phase.

Transfer is a more sophisticated method of translating between a language pair. The dictionary-based system described above often produces ungrammatical results, not to mention problems with semantics. If the words of a phrase are translated without the context of the whole sentence, the translation is not

guaranteed to get all the dependencies right, and the constituents will not be inflected correctly in the translation.

The intermediate representation of a transfer-based system extends to syntax and might include semantic or pragmatic features, for example word sense disambiguation. Such system could even have very sophisticated rules for handling the fine nuances, such as animacy or politeness, and it would know not to translate idiomatic expressions literally. The essential difference between transfer and interlingua is that a transfer-based system is always specific to one language pair; in effect, it is still a bilingual dictionary, although more elaborate. Many aspects of the representation are determined by the two languages of a pair: the rule set for a pair of closely related languages is very different from the rule set of two completely unrelated languages.

The interlingual system ranks the highest in the intermediate representation: it has to be abstract enough to be language independent. With an interlingua, one could translate between any two languages, as long as both have been mapped to the interlingua. Of course, the interlingual system has to include morphological and syntactic analysis and generation, in order to be used for any real language. The interlingua itself can be a human-readable, real or constructed language—past examples include Esperanto (Witkam, 2006)—or an abstract notation system. An abstract version of the interlingua could consist of data structures that denote the syntactic and semantic properties of the utterances, not just the properties relevant to one language pair, but enough properties to cover all the languages that one wants to include in the translation system.

In a broad sense, any pivot language can be considered an interlingua. Suppose we have the languages `L1`, `L2`, `L3`, `L4` and `IL`, and create the pairs `L{1-4}`–`IL` and `IL`–`L{1-4}` using transfer. According to the broad definition, the language `IL` is an interlingua. Texts can be translated between any two languages `L{1-4}`, without defining rules for every possible pair. In the strict sense, in order to be considered interlingual, the translation process should consist of analysis and generation, rather than transformation of syntactic structures. As for the scenario with `L{1-4}` and `IL`, `IL` is definitely a pivot language, but because of the manner of the mapping between `L{1-4}` and `IL`, it cannot be called an interlingua in the strict sense.

Constructing a truly universal interlingua is, in practice, an impossible task; see Madsen (2009) for a thorough analysis on the limitations of machine translation. It would mean constructing a system that is prepared for everything that a human could possibly utter, and a mapping from that representation to all possible linguistic systems—it would have to have the information of all possible ways in which a meaning could be mapped to an utterance. However, interlingual approach has been used succesfully in certain applications, with a specific domain, a relatively small lexicon and possibly syntactically restricted input text.

Statistical machine translation We present the statistical machine translation (SMT) paradigm to give context to MOLTO. The modern machine translation industry is predominantly based on the statistical method. SMT has many advantages, especially for the needs of a casual user: it requires less human work and it is more robust, which means that the user will get at least some kind of output, enough to get an idea of the content. This level of accuracy is called browsing quality, as opposed to publishing quality, which is the aim of MOLTO. This section is just a quick overview of statistical machine translation; for more information, see for example Lopez (2008).

In statistical NLP, language is modelled in terms of probabilities of word sequences. Probabilities of translation equivalents are computed from bilingual aligned corpora: given n-1 previous words in the source text, what is the probability of the nth word w_s being translated as w_t? We express this probability with $P(w_t|w_s)$. For an example word *saw*, for the language pair English–Spanish and with n=3, the probablity $P(vio|saw)$ should be higher in a context such as *then she saw*, and $P(sierra|saw)$ in *with a saw*. Modern SMT systems use sequences of varying lengths as the units to be aligned, but the basic idea is the same: a sequence of 1 to n words in the source language corresponds to a sequence of 1 to n words in the target language, and the probability of a translation given the source depends on the context provided by other words. Possible reordering of the translation is handled with a language model of the target language.

The quality of translation depends on many things, such as the languages in question, the coverage of the corpora and the size of n. Translating to a language with little morphology is easier than to a language with rich morphology;

translating between two similar languages is easier than between very different languages. Common utterances in general translate well: a fixed phrase like *I miss you* is frequently attested, and a statistical system can give very idiomatic results in different languages. However, if the phrase is part of a more complex sentence, or has less typical persons, such as *you miss me*, the system has no way to connect that to the frequently seen phrase, and the result can be poor (Ranta, 2011b).

Statistical machine translation might seem imprecise or crude, but the trend of recent years has been to develop a hybrid system, combining the best parts of rule-based and statistical methods. In MOLTO, we have experimented with this too, for example, by enhancing the bilingual corpora with alignments generated by our grammars (España, 2011). This helps to solve the problem with sparse data: by exhaustive generation we will get less common utterances in the phrase tables, which makes the translation of *you miss me* as reliable as *I miss you*.

2.1.3 Ambiguity in the context of machine translation

A natural language phrase can be ambiguous lexically or syntactically. Sometimes a third term, semantic ambiguity, is used to describe sentences that are vague and open to interpretation, such that their meanings are not clearly defined or agreed on. For example, the words describing size cover a very wide range of absolute sizes depending on the context; a big mouse is always smaller than a small elephant, and a big house may have different criteria depending on the culture. Given that these descriptions are hard even for humans, we exclude semantic ambiguity or vagueness and concentrate on lexical and syntactic ambiguity, with the assumption that an ambiguous sentence has multiple readings, but all possibilities can be enumerated, and the correct option for each context is necessarily among them.

Lexical ambiguity is caused by homonymy or polysemy, that is, a single word form has multiple meanings. If a sentence is lexically ambiguous, its syntactic structure is clear, but it contains one or more homonymous words, and the meaning of the whole sentence depends on the meaning of the homonymous word. Consider the example *I put 100 euros in the bank*. The sentence can refer to depositing the money to a bank account, or burying the money in

the mud on the edge of a river; the latter interpretation is unlikely for anyone with world knowledge, but still technically possible.

Syntactic ambiguity means that a phrase has more than one possible syntax tree. This can be due to homonymous words; especially function words like prepositions or conjunctions, or content words whose homonym is in different class, e.g. *duck* 'aquatic bird' or 'to lower'. The scope of operators can also cause ambiguity, like in the NP *old men and women*, where it is unclear whether the women are old too, or *teachers of French and German*, which can refer to a set of teachers whose each member teaches both of the languages, or a set of teachers whose each member teaches only one of the languages. A syntactically ambiguous phrase need not have distinct truth values for each syntax tree: for example, the NP *young boys and girls* has the same two parses as the version with their older counterparts; *young* might modify only *boys* or *boys and girls*, but in either case, both boys and girls are young by definition.

When we say that natural languages are ambiguous, we need to specify in relation to what. The objective of machine translation is to provide a mapping between two or more representations of some kind of reality; it is a problem of artificial intelligence to act in a meaningful way in that reality. In machine translation, problems arise only when the source language has less elaborate distinctions than the target language, and the translation requires information that the original does not state explicitly.

For example, the Finnish personal pronouns do not differentiate between gender, and nouns and noun phrases do not have obligatory definiteness marking, such as articles. When a Finnish text is translated into a language with the same features, the readers of the translated text get the same information at the same level of ambiguity, and they will do the same reasoning as the readers of the original sentence. But if the text is translated into a language that requires such distinctions, the translator has to make a choice, which could be different from what the author meant.

Human translators normally translate documents considering a complete paragraph or section, and they might do paraphrasing, such as altering the order in which the information is presented or omitting some details that are irrelevant to the intended audience. Many MT systems have a *translation unit* of one sentence or even shorter, and it is not possible to use context

outside of it. In many cases, a translation unit of one sentence is enough: it is syntactically complete, that is, all syntactic dependencies and agreements can be caught when translating complete sentences. Translating each sentence is a secure way to get all information through; the translation system is not meant to judge which information is important, it just translates everything, preserving the structure of the document. For that purpose, translation unit of one sentence is justified. As for ambiguity solving, most of the techniques work with a window of one sentence. Means for solving lexical ambiguity include looking for words nearby; often they are found within the sentence. In cases of syntactic ambiguity, the context of other sentences does not solve that, unless the parses have different meanings, and the environment provides semantic cues to determine the right parse. Thus extending a translation unit to a paragraph or document would make a difference in few cases; most of the time it would just increase the complexity of the system.

2.2 Ontologies

In this section we discuss ontologies. First we introduce some basic concepts, in section 2.2.2 we talk about the sources of ontologies and in section 2.2.3 we present some uses of ontologies.

2.2.1 Basic concepts

In information science, *ontology* means a collection of concepts and relationships between them. The relationships include things like category membership ("Garfield is a pet", "Wading is a form of movement") and properties of the instances ("Garfield is fat") and categories ("Pets have owners")—from the structure it is possible to infer that Garfield has an owner, even if it is not explicitly stated. Ontologies have a hierarchical structure, starting from a generic superclass, such as `Thing` or `Entity`, to more specific subclasses. An ontology may be domain-specific, with very detailed descriptions, or it can be general and contain broader statements. These general ontologies, called *upper ontologies*, can be used to make different domain ontologies interoperable.

As for technical details, the examples are in a format called Resource Description Framework (RDF), where the information is presented as triples

of subject, predicate and object. One triple describes one thing about the subject, such as name, superclass, or more domain-specific details: for a person, important details might include age, family relations and job; for art objects, the artist, current location, size of the object and such.

Another important concept is Uniform Resource Identifier (URI), defined by Berners-Lee et al. (2005). Anything that can be identified by a URI is called a resource. They can be concrete documents available in the web or abstract representations: a URI of an image can be a fully functional address (URL) that points to the actual data, but for an entity such as person or abstract concept, a URI is just a name that identifies and represents it. Predicates of the triples are like any other item: a concept such as *age* is represented by a URI, and that URI can be used in a triple whose subject is a person and object is a number. An item can be a subject in one triple and a predicate or an object in other. The object of a triple can be a URI or a literal, such as an integer or plain text, but subject and predicate must be URIs. Below is an excerpt from an ontology entry of Mona Lisa.

```
@prefix dbpedia: <http://dbpedia.org/resource/> .
@prefix dbp-prop: <http://dbpedia.org/property/> .

dbpedia:Mona_Lisa  dbp-prop:artist  dbpedia:Leonardo_da_Vinci .
dbpedia:Mona_Lisa  dbp-prop:museum  dbpedia:Louvre .
dbpedia:Mona_Lisa  dbp-prop:name    "Mona Lisa"@en .
dbpedia:Mona_Lisa  dbp-prop:name    "La Gioconda"@it .
```

The notation `prefix` is a way to shorten the URIs; in full form the URI for Mona Lisa would be `http://dbpedia.org/resource/Mona_Lisa`, and for name `http://dbpedia.org/property/name`. The objects of the latter two triples do not have a URI for them, they are string literals and denote the name of the object in English and Italian.

2.2.2 Ontology sources

The unchallenged assumption in this thesis is that we have an ontology in which natural language ambiguities and different meaning clusters across languages are resolved, and all we need to do is to specify a domain to get a list of

well-defined concepts in multiple languages. The other end of text-ontology interoperability is the creation of the ontologies: either building manually or learning automatically from free or semi-structured text. *Ontology acquisition* is a subfield of information extraction, and is outside the scope of this study; for an overview see e.g. Buitelaar et al. (2005).

There is a lot of information in free text, that is, collected by humans and meant for humans, not structured in a machine-readable way. In addition to free text, there are semi-structured data collections, such as Wikipedia. Instead of building ontologies manually, they can be *learned* from these sources—for example the ontology DBpedia gets its content from Wikipedia articles. The subclass and superclass relations are derived from the categories, and other relations, such as date of birth for humans, come from Wikipedia's infoboxes, which have a fixed format. The translations for the concepts come from the different language versions of the articles. In addition, the data is linked to other free data sources on the web, as a part of the Linked Data initiative (Bizer et al., 2009).

Seppo Nyrkkö's doctoral thesis, also a part of MOLTO, explores further aspects of text and ontology. Nyrkkö uses various types of resources to unify the information from many sources; for example, the same topic can be covered in a general ontology, a specialised ontology and free text, and Nyrkkö uses both statistical and rule-based methods to compute similarities between the concepts. The study is presented in more detail in section 5, along with other related work.

2.2.3 Ontology uses

A concrete motivation for well-structured, machine-readable way of representing information is the fact that there is so much information in the world. *Linked Data*, described in Bizer et al. (2009), is a model of structuring and connecting the data on the web. The hypertext web links documents to each other—the unit of information is a web page, and the links do not have any finer semantics. Bizer et al. (2009) make a distinction between *untyped* and *typed hyperlinks*; a simple hyperlink from a document to another is untyped, while an RDF link is typed, because the predicate of the triple expresses the type of the connection between the subject and the object.

In a hypertext web, a page describing an artist might include a link to a page describing a song she has performed. If the page about the song is very informative, one can find there a list of other performers, and get a complete list of everyone who has ever performed that song. In a linked data model, artists and songs are resources, and can be linked to each other by different types of relations, one of which is `performs` or, conversely, `performedBy`. If someone—human or computer—browsing the linked data graph wants to know all songs performed by a certain artist or all performers of a certain song, the information can be retrieved easily, without anyone having to manually gather and update a separate document describing just that. Information is distributed on different parts of the web and can be dynamically fetched when needed. This technology forms the basis of automatic question answering. The difference between a document search engine, such as Google, and a question answering system, such as Wolfram Alpha is that the former type of engine returns documents that contain the search terms. The latter type returns answers that are not necessarily presented in that exact form in any single document. By traversing the links between the data, a question answering system can find, process and present information that answers the user's query.

2.3 Human-computer collaboration

Labour can be divided between human and computer according to different models. In this section, we look at two ways for humans and computers to collaborate.

The term *Web 2.0*, first used in 2005 (O'Reilly, 2007), describes the WWW as a platform of collaborative work; anyone can be both a publisher and a consumer of information. The "version number" 2.0 does not mean a concrete change in any core technologies, but is called that as opposed to the earlier stages of the WWW, which resembled traditional publishing. *Crowdsourcing* is another phenomenon that blurs the line between experts and amateurs. An example of crowdsourcing is providing different language versions of a website with the help of its users instead of hiring professional translators for the task.

For *Web 3.0*, no one clear definition exists, but one definition focuses on the type of collaboration between human and computer. Whereas work typical of Web 2.0 is done by humans for humans, in Web 3.0 a few experts do work

initially directed to computers, for example creating ontologies. In this model, information is in a form that is of little use for a non-specialist, but ultimately representing a taxonomy in a machine-readable format will shift a lot of tedious work from the human to the computer.

In the domain of multilingual websites, translation as a community project is the first approach. Almost anyone who knows the language can contribute, and when there are many people, everyone's workload is small. There is constant but minimal work: whenever the website updates, it is easy to get someone from an active user base to translate the changes. The collaboration model employed in MOLTO belongs to the latter approach. The initial effort of creating multilingual grammars for the domain is big, but after that, there is no need to retranslate; updates to whichever document are generated simultaneously into all language versions. This process can be described as multilingual content creation in addition to machine translation. The technical details of the grammars will be explained in the next chapter, section 3.1.

3 Resources

This chapter introduces the resources that the MOLTO project offers. Unlike the previous chapter, which discussed general theoretical background, this chapter refers to specific programming languages, techniques and data collections that are available for the task of lexicon management. This is not a list of everything included in MOLTO, just the resources that are relevant to this study.

In section 3.1, we introduce the programming language Grammatical Framework, which is used to write the multilingual grammars. The data collection FactForge, which combines many ontologies and data sets, is presented in section 3.2. The other resources are the terminology platform TermFactory (section 3.3) and the lexical database WordNet (section 3.4).

3.1 Grammatical Framework

Grammatical Framework (GF) is a grammar formalism and a typed functional programming language, specialised in representing multilingual grammars. A

GF grammar includes an abstract syntax and one or more concrete syntaxes. The abstract syntax declares the kinds of concepts and utterances that can be expressed in the grammar, and in a concrete syntax they are given representations in some real language. The mapping between the abstract and the concrete syntax is two-way: the direction from concrete syntax to abstract syntax is called *parsing*, and the direction from abstract to concrete is called *linearisation*.

In the context of multilingual translation, the abstract syntax functions as an interlingua. The translation process consists of parsing from one language to the abstract syntax and linearisation from the abstract syntax to all other languages. The abstract syntax tree describes semantics; what gets translated is the meaning, not the syntactic structure. Language-dependent elements are handled in the concrete syntax of each language. Abstract syntax is discussed in detail in section 3.1.2 and concrete syntax in 3.1.3.

3.1.1 Functional programming basics

The theoretical background of GF is in functional programming and type theory. GF is a typed language: all values have a type, and the compiler performs a type check to ensure that the functions operate on appropriately typed data, as a way to prevent programming errors. In the functional programming paradigm, computation is done by evaluating functions, not by changes of state. There are no global variables or reassignment; the only variables are function internal. All values are produced by evaluating a function or a chain of functions, in which the result of each function is passed on as an argument to the next function in the chain. Functions can be passed as arguments and returned by other functions, and function literals can be formed. All entities with these properties are called *first-class objects*, and in GF this includes functions; primitive data types, such as integers and strings; and composite data types, such as records and associative tables.

A function has a *return type*, which is the type of the first-class object returned by the function, and a *type signature*, which means the types of the arguments and the return type of the function. A function that takes as its arguments two integers and returns an integer has a return type `Integer` and a type signature of `Integer` → `Integer` → `Integer`. Any first-class object

can be an argument or a return value: for example, a type signature `Integer → {s: String, i: Integer} → (Integer → String → String)` denotes a function which takes an integer and a record as its arguments, and returns a function. The returned `(Integer → String → String)` is the type signature of the constructed function, and the return type of the constructing function.

A constant is an argumentless function: it only returns a value, so its type signature is the same as its return type. Argument-taking functions take a value and return a new value. In GF grammars, the atomic lexical items in the language—nouns, adjectives, verbs, adverbs, conjunctions—are implemented as argumentless functions, called *lexical functions*. The functions with arguments, *phrase-building functions*, take lexical items as their arguments and produce more complex structures. However, there is no formal distinction between the two kinds of functions, we introduce the terms mainly for the human reader's sake.

3.1.2 Abstract syntax

The abstract syntax consists of category[1] declarations and function declarations. Let us now look at an example grammar, which, for simplicity, allows only noun phrases. The categories of the grammar are `Determiner`, `Quality`, `Kind` and `Item`. Every complete construction in this grammar will belong to the category `Item`, making it analogous to the start symbol in context-free grammars. Program 1 shows a complete abstract syntax of a GF grammar.

The grammar has six lexical functions: `This`, `These`, `Big`, `Red`, `House` and `Dog`. On the left side of the colon is the name and on the right is the return type of the function. The lexical functions are analogous to the terminal symbols in context-free grammars. There is no lexical function that would just return an `Item`, just as in context-free grammars the start symbol is not usually followed by terminal symbols.

In our grammar, we want an `Item` to be composed of one `Determiner`, one `Kind` and zero or more instances of `Quality`. The phrase-building function `Mod` takes two arguments: one of type `Quality`, another of type `Kind`, and

[1] As for terminology, we will use the terms *category* and *type* interchangeably. Saying that a value belongs to a category `X` means the same that the type of the value is `X`.

Program 1 The abstract syntax of a noun phrase grammar

```
abstract NounPhraseGrammar = {
  flags startcat = Item ;

  cat
    Determiner ;
    Quality ;
    Kind ;
    Item ;

  fun
    -- lexical functions
    This  : Determiner ;
    These : Determiner ;
    Big   : Quality ;
    Red   : Quality ;
    House : Kind ;
    Dog   : Kind ;

    -- phrase-building functions
    Mod : Quality    -> Kind -> Kind ;
    Det : Determiner -> Kind -> Item ;
}
```

constructs a new element of type `Kind` out of them. `Det` takes a `Determiner` and a `Kind` and produces an `Item`. Valid abstract syntax expressions in this grammar are shown in figures 1 and 2.

Figure 1: Det This Dog

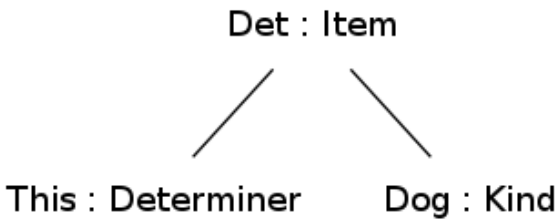
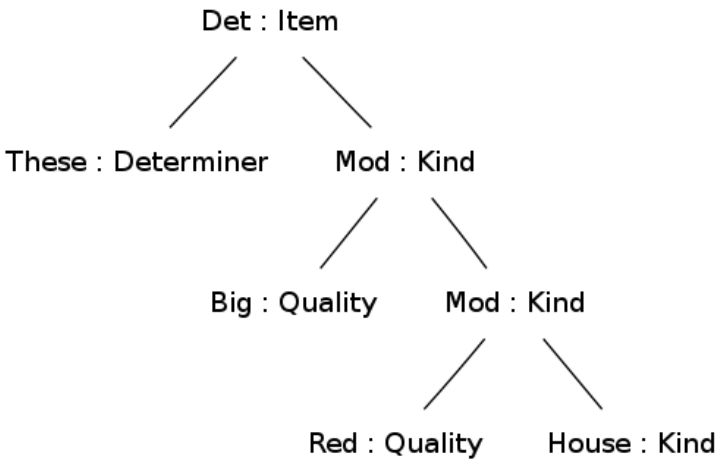


Figure 2: Det These (Mod Big (Mod Red House))



In the first example, the function `Det` is applied to the values returned by the lexical functions `This` and `Dog`. The second example demonstrates the function `Mod`, which adds a `Quality` to a `Kind`. The brackets show the application order of the functions: first the innermost expression `Mod Red House` is evaluated into an element of type `Kind`. Then the next `Mod` is applied to the `Quality` returned by `Big` and to the `Kind` returned by `Mod Red House`. Finally, the function `Det` is applied to the determiner `These` and to the result of the two `Mod` applications, and the final result is an element of type `Item`.

By looking at the abstract syntax one cannot know how the functions are linearised in the concrete syntax. In one language, adding a `Quality` to a `Kind` might be a matter of trivial concatenation, whereas in some other language

it might have complicated rules, including agreement in number, gender and case. On the one hand, writing the abstract syntax is easy, since the grammar writer need not know the implementation details of the functions in various languages. On the other hand, that is exactly what makes the abstract syntax writing hard: it really needs to be abstract. In the earlier example, we have deliberately used non-grammatical terms; *quality* instead of *adjective*, *kind* instead of *noun*. Such constructions may not exist in all languages, so it is better to use common descriptions instead of grammatical categories of a specific language.

3.1.3 Concrete syntax

When we build a concrete syntax for some language, we have to give both categories and functions concrete definitions. The categories are linearised as data structures, and the lexical functions have these data structures, with appropriate data, as their return values. The phrase-building functions are implemented for each language in a different way, to handle the data structures specific to that language.

The kinds of data structures needed depend on language. Even in our small noun phrase grammar with no verb morphology nor adjective comparation, we have to consider the inflection and the agreement of nominals. English only requires number agreement (with determiners and nouns, not with adjectives), in Spanish there is also gender agreement and in Swedish, in addition to the two, definiteness agreement: the definiteness of a noun phrase is marked in all constituents, whereas in English only the determiner carries that information.

Let us look at the categories `Kind` and `Quality`. In English concrete syntax, the data structure for `Kind` needs a field with two options, one for singular and one for plural. `Quality` needs only one option, because there is no number distinction in adjectives. In Spanish, both nouns and adjectives have separate singular and plural forms. Nouns have a fixed gender and adjectives have two gender forms, from which one is chosen determined by the gender of the noun they modify. Thus in Spanish concrete syntax, `Kind` needs a field for the noun, with singular and plural forms, and an additional field for expressing

Program 2 The concrete syntax of a noun phrase grammar

```
concrete NPEnglish of NounPhraseGrammar= {

  param Number = Sg | Pl ;

  lincat
    Determiner = {s: Str ; num: Number} ;
    Quality = {s: Str} ;
    Kind = {s: Number => Str} ;
    Item = {s: Str} ;

  lin
    -- the lexical functions
    This  = {s = "this" ; num = Sg} ;
    These = {s = "these" ; num = Pl} ;
    Big   = {s = "big"} ;
    Red   = {s = "red"} ;
    House = {s = table {Sg => "house" ; Pl => "houses"}} ;
    Dog   = {s = table {Sg => "dog" ; Pl => "dogs"}} ;

    -- the phrase-building functions
    Det det  kind = {s = det.s ++ kind.s ! det.num} ;
    Mod qual kind = {s = table {Sg => qual.s ++ kind.s ! Sg ;
                                Pl => qual.s ++ kind.s ! Pl}} ;
}
```

the gender of the noun, and the `s` field in `Quality` includes 4 options, for all combinations of {singular, plural} and {masculine, feminine}.

When the categories are defined, we can start writing the concrete syntax functions. Defining the lexical functions means basically filling the slots: we can linearise `House` by typing the strings *house* and *houses* to the correct fields in the data structure that is common to all `Kinds`, and have `House` to return that value. The phrase-building functions are more complicated, since they take arguments and perform operations on them. A `Mod` in English does only concatenation: it takes the only string value of the `Quality` and the two options of the `Kind`, and constructs a new `Kind`, whose fields will have the combined values. The Spanish `Mod` has to select a correct gender, since `Quality` has two gender options, but `Kind` has a fixed gender.

In our grammar, a value of type `Determiner` is the element that completes a `Kind` into an `Item`. This means that the incomplete noun phrase `Kind` has some of its options open. The start category of the grammar, `Item`, always has only one field: it has all its constituents in the right forms, the combination determined by evaluating the `Item`-forming expression from the bottom to the top, starting from multiple possibilities in the atomic categories and dropping impossible ones along the way.

As for English, when `Mod Red House` is evaluated, the resulting `Kind` contains the combinations *red house* and *red houses*. The number options of `Kind` are open, and the category `Determiner` must have an inherent number in order to choose the matching value in `Kind`.[2] The example grammar has inherently singular and plural determiners `This` and `These`, which include the string representation of the word and the parameter that is used to choose the right value from the `Kind`. This way, the linearisations of the following expressions will have the right agreement.

```
Det This  (Mod Red House) ===> "this red house"
Det These (Mod Red House) ===> "these red houses"
```

The decision that a determiner completes a common noun (CN) into a noun phrase comes from a tradition of logical semantics (Ranta, 2011a). The

[2]This is implemented with an associative table, with the number as the key and the string combination as the value. The number belongs to a *parameter type*, which is used only as a table key. In GF syntax: `table {Sg => ‘‘red house’’ ; Pl => ‘‘red houses’’}`.

idea is that a CN is a description and with a determiner it is quantified. From a linguistic point of view, this is not always the best option: determiners do not necessarily include number in all languages, or they are not obligatory to form a noun phrase, or there might be more than two number categories. However, all this can be handled, because GF itself is not based on any particular syntactic theory, and it gives a lot of freedom with respect to implementing different linguistic features.

3.1.4 Resource grammars and application grammars

Writing a large-scale GF application would be really tedious, if syntactic and morphological functions had to be implemented separately for every language in every application. For example, the GF grammars for the mathematics case study and the museum case study seem different: the structure and the types of the abstract syntax reflect in each grammar the distinctions that are important in that particular domain. In terms of abstract syntax, the two grammars have little in common. However, if we look at the concrete syntax level, the Finnish concrete syntax of a mathematics grammar inevitably shares a lot with the Finnish concrete syntax of a museum grammar. The syntax and morphology of a language do not change according to the domain: whether we talk about mathematics or museum objects, there is no difference in the way that nouns inflect or a modifier attaches to a head. This is why syntax and morphology are moved to a module, called *resource grammar*, that can be shared by many different applications. Like any software library, it is imported to the application grammar, where its functions can be used without knowing how they are implemented.

The resource grammar is a GF grammar that implements the basic syntax and morphology of a language, including also a core lexicon of around 350 words. As of November 2012, 25 languages have a resource grammar, and they are all freely available as open-source libraries. Each resource grammar consists of two parts: one which includes the categories and operations common to all languages in the resource grammar library, and other which includes syntactic structures specific to each language. For the language-specific part, each language has its own abstract syntax and concrete syntax. The common part has a common abstract syntax and 25 concrete syntaxes.

The categories of the resource grammar are linguistic categories, like adjective, verb phrase or relative clause. The resource grammar functions operate on the categories. Program 4 shows some function declarations from the abstract syntax of the resource grammar. The type `CN` is a common noun, it includes an obligatory noun and arbitrarily many modifiers. The types `AP`, `RS` and `Adv`, respectively adjective phrase, relative clause and adverb, are allowed to modify the `CN`, making the result still a `CN`. The type `Det`, determiner, can complete a `CN` into a noun phrase.

Program 3 Fragment of the abstract syntax of a resource grammar

```
fun
  AdjCN : AP -> CN  -> CN ;   -- big house
  RelCN : CN -> RS  -> CN ;   -- house that John bought
  AdvCN : CN -> Adv -> CN ;   -- house on the hill
  DetCN : Det -> CN -> NP ;   -- the man
```

The fraction is from the common part of the resource grammar's abstract syntax. What is declared here is that all languages should have nouns and adjectives, and they should be able to combine. The functions are implemented for each language in their own concrete syntax. For the end user there is an API with consistent function names, such as `mkN`, `mkCN`, `mkVP`, which are overloaded with the many possibilities on how to construct a `CN`. The user just needs to write `mkCN <arguments>` to construct any type of `CN`, without having to specify whether it is an `AdjCN` or `RelCN`.

Now, what is left for the application grammar developer to do, if she need not write rules about how a modifier attaches to the head? In our toy grammar not much, aside from defining the lexicon, but in a more elaborate grammar there are semantic and pragmatic issues to consider. Below is an example from the abstract syntax of a tourist phrasebook grammar.

Program 4 Fragment of the abstract syntax of an application grammar

```
fun
  AHaveCurr : Person -> Currency    -> Action ;  -- you have dollars
  ACitizen  : Person -> Citizenship -> Action ;  -- you are Swedish
  ABePlace  : Person -> Place       -> Action ;  -- you are in the bar
```

The functions are very specific, compared to those of a resource grammar, and the categories are not grammatical categories, but semantic descriptions. In the resource grammar it would not make sense to have a specific category for currency or citizenship, but in an application grammar it could be an important distinction, with respect to style or other pragmatic issues.

The phrases are constructed in their respective concrete syntaxes with any resource grammar operations that the grammar writer chooses. For example, `AHaveCurr` is a function in the abstract syntax of an application grammar. It is linearised in the English concrete syntax using the English resource grammar, and in the Finnish concrete syntax using the Finnish resource grammar. There is no need for the English and Finnish linearisations to have an identical syntactic structure, in terms of linguistics, yet they both map to the same abstract syntax tree, and thus are each other's translations. This allows the sentences to sound more natural and idiomatic in each language. An abstract syntax category like `VerbalAttribute` could be linearised in some language as a relative clause and in another language as a gerund. Or a phrase-building function `ALikesB` could be linearised in English with A as the subject and B as the direct object, *A likes B*, and in Spanish with B as the subject and A as the indirect object, literally *B pleases A*.

3.1.5 Semantic restrictions in GF

Given GF's theoretical background, it is possible to express conditions of semantic well-formedness in the grammar. The resource grammar operates with types such as noun phrase, transitive verb or conjunction, and a VP-making function could take any `V2` and `NP` and construct a `VP` out of them, regardless of their meanings. For expressing semantic restrictions in an application grammar, we can use in the abstract syntax more elaborate types. The phrasebook example, `AHaveCurr : Person → Currency → Action`, guarantees that we cannot apply the function `AHaveCurr` to any two noun phrases, rather, they have to be noun phrases whose abstract syntax types are `Person` and `Currency`. Even more expressivity is gained by the use of *dependent types*, such that it is possible to represent a complete ontology in GF. We will present these methods in the following subsections.

Dependent types The distinctions such as separating humans from objects are broad enough to have their own categories. But sometimes we need even finer distinctions, which may only apply to certain constructions. For example, in the domain of museum object descriptions, the people appearing in the texts are likely to be either artists or subjects. As a concrete motivation for fine-tuning the semantic distinctions in a grammar, we examined the corpus provided by Göteborgs Stadsmuseum, and found out that the descriptions *artwork of* (depicting) and *artwork by* (authored) are often expressed with the same construction, using the Swedish preposition *av* for both meanings. Without any context, an expression like *ett porträtt av Mona Lisa* is ambiguous as to whether Mona Lisa is the subject or the painter.

As a solution, we could split the abstract syntax category `Person` into two categories, `Painter` and `Subject`, and define the functions `authoredBy` and `depicting` so that they accept only one as their argument. This way, even if the constructions are identical, the type of the argument disambiguates the expression. Program 5 shows this approach.

Program 5 Restrictive abstract syntax, without dependent types

```
cat
  Painting ;
  Painter ;
  Subject ;
fun
  portrait   : Painting ;
  authoredBy : Painting -> Painter -> Painting ;
  depicting  : Painting -> Subject -> Painting ;
  Mona_Lisa  : Subject ;
  Leonardo   : Painter ;
```

However, separating painters and subjects would prove too restrictive: after all, there are many portraits depicting painters, but the grammar would not allow a painter to be in any other role than an author—*a portrait of Leonardo da Vinci* would be considered a syntax error and not be parsed. Furthermore, if there are functions common to all people, such as being born or dying, each of them would have to be defined separately for painters and subjects. There are some actions that can only be done to one of them, but

since the majority of the actions are the same, it may not be feasible to put them in different categories.

What can be done, instead, is to define dependent types. In addition to the simple type `Person`, we add dependent types `Painter` and `Subject`, which take `Person` as argument, producing the types `Painter Person` and `Subject Person`. They provide a semantic layer on top of the syntactic level, more flexible and easier to modify. Program 6 shows the same excerpt from a museum grammar, with dependent types:

Program 6 Restrictive abstract syntax with dependent types

```
cat
  Painting ;
  Person ;
  Subject Person ;
  Painter Person ;
fun
  Mona_Lisa  : Person ;
  Leonardo   : Person ;
  portrait   : Painting ;
  authoredBy : Painting -> (p : Person) -> Painter p -> Painting ;
  depicting  : Painting -> (p : Person) -> Subject p -> Painting ;

  -- proof objects
  Mona_Lisa_Subject : Subject Mona_Lisa ;
  Leonardo_Painter  : Painter Leonardo ;
```

The declarations of portrait, Mona Lisa and Leonardo da Vinci are not different from what we have seen before. Let us look at the functions `authoredBy` and `depicting`. Like before, they take a `Painting` and a `Person` as their arguments and construct a new value of type `Painting`. The notation `(p : Person)` is a way to bind a variable to the argument type, so that we can refer to the same argument later. The third argument is of the dependent type `Painter p` or `Subject p`, in which the variable `p` must be the `Person` given as the second argument. The last two functions in the grammar assert which combinations of persons and roles are valid. They are called *proof objects*; technically they are (argumentless) functions just like any other under

the `fun` title, but their purpose is to control that the function's arguments are semantically valid, and they are not linearised in the concrete syntax.

It is possible to define many proof objects for one element. Say we want to add a self-portrait into the lexicon, and require that an author of a self-portrait can only be a person who is both a subject and a painter. To do this, we would add to the abstract syntax a function that constructs the self-portrait, and accompanying proof objects for each painter who has done a self-portrait. Below is an example for Leonardo da Vinci.

Program 7 Semantic requirements for the author/model of a self-portrait

```
fun
  selfportrait : Painting -> (p : Person) ->
                 Painter p -> Subject p -> Painting ;
  Leonardo_Subject : Subject Leonardo ;
```

To sum it up, the function `authoredBy` can be applied to any painting, and any person `p`, for which there is a proof object `Painter p` defined. Adding dependent types does not rule out anything previously done. The basic type of all people is still `Person`. In addition, Leonardo da Vinci belongs to the categories `Painter Person` and `Subject Person`, and Mona Lisa to `Subject Person`. If some language treats the owner of a portrait in a way that results in ambiguity, it is easy to add a new type, `Collector Person`, and write a proof object for each person who is a collector.

GF and ontologies So far our concern has been to handle the ambiguity of natural language, in this case the same preposition being used for author and subject. This kind of solution is not enough to guarantee semantic well-formedness in all aspects—for instance, the function `selfportrait` does not determine that a painting is actually painted by Leonardo, rather, it only checks that Leonardo is a possible choice for an author of a self-portrait. But we could still claim that any portrait is a self-portrait of Leonardo: the function only checks whether the potential author has ever been a subject, nothing about the painting itself.

To ensure that the painting really is a self-portrait, we need a more complex structure. Dependent types can take more than one argument: for all

requirements of a self-portrait, we need a type constructor that takes both `Person` and `Painting`.

Program 8 Complete semantic requirements for a self-portrait

```
cat
  Person ;
  Painting ;
  Authored Person Painting ;
  Depicting Person Painting ;
fun
  Portrait_of_a_man : Painting ;
  Mona_Lisa : Painting
  Leonardo : Person ;
  depicting : (pe : Person) -> (pa: Painting) ->
    Depicting pe pa -> Painting ;
  authoredBy : (pe : Person) -> (pa: Painting) ->
    Authored pe pa -> Painting ;
  selfportrait : (pe : Person) -> (pa: Painting) ->
    Authored pe pa -> Depicting pe pa -> Painting ;
  Authored_Leonardo_ML : Authored Leonardo Mona_Lisa ;
  Authored_Leonardo_PAM : Authored Leonardo Portrait_of_a_man ;
  Depicting_Leonardo_PAM : Depicting Leonardo Portrait_of_a_man ;
```

Doing this manually would be a demanding task. Even in the toy example above, the function declarations get quite heavy. The grammar is starting to look like an ontology: it does not just give us syntactic rules on how to say something about paintings and subjects, but it requires that what we say is based in reality. Thus it should be no surprise that an ontology can be translated to GF (Enache, 2010). GF's support for dependent types, which was used for fine semantic distinctions in the previous section, can be used to represent the whole inheritance hierarchy of an ontology. Reasoning is also done in the type system.

Classes are represented as values of type `Class`, and statements have value of some type `DepType Class (Class)+`. For example, `Painting` and `Artefact` are values of type `Class`, and a rule saying that painting is a kind of artefact is of type `Inherits Painting Artefact`. There are inference rules

taking care of e.g. transitivity and self-inheritance; for the latter property the rule is a function of type `(c : Class) -> Inherits c c`. Instances are also covered by the type system: Mona Lisa's immediate superclass is Painting, so it is of type `Ind Painting`, and it can be coerced to indirect instance of Painting's superclass, such as `Artefact` or any of its superclasses.

However, GF is not optimised for doing ontology reasoning (p. 35); most of the work done in this area has been proof of concept, not an attempt to design a large-scale system. We will not cover the topic any further in this study, for more information see Enache (2010) and Angelov and Enache (2011).

Even without elaborate semantic restrictions, ontologies could prove useful simply as the source of the lexicon. Ontologies are based on concepts, not words, so they are a good source of abstract syntax lexical elements. Next we will see the ontologies that are used as a resource in this project.

3.2 FactForge

FactForge is a collection of data sets and ontologies. It is hosted by Ontotext, a company specialised in semantic technology. It consists of 15 open data sets and ontologies, gathered all around the web, whose items are linked to each other: for example, an entry about a singer in a general ontology would be linked to her discography in a music ontology. In technical terms, the URI representing the singer appears in the general ontology in certain triples, such as name, date of birth and nationality. In a more specialised music ontology, the singer might be represented by a different URI and appear in different triples, such as vocal range. By interlinking the entries, a user can do a query on all ontologies and get all facts on the singer, even if they are originally from different sources. The number of all statements in FactForge, explicit and inferred, is over 2 billion (Damova, 2012).

FactForge is a useful source for MOLTO, because the terms often have translations in many languages. However, there is no morphological or syntactic information, aside from part of speech information provided by WordNet, which is one of the datasets. As for more general details, FactForge is free to use, has wide coverage of facts from different domains and is updated approximately once per month. The main purpose of FactForge in this study is to

provide terms, as described in section 4.2; in addition we talk about a natural language query tool for FactForge in section 4.2.1.

3.3 TermFactory

TermFactory (TF) is a platform and a collection of tools for collaborative multilingual term management. The style of work is a combination of Web 2.0 and 3.0: people are the ones doing the work, but in a structured way, using a specific machine-readable format. In addition, an expert community is responsible for reviewing the quality of the work. Translations can be confirmed or rejected, and new translations can be added. The result is a term collection whose entries include a variety of information that can be used in NLP applications, such as inflection paradigm and valency. There are more immediate benefits for the users, such as collaboration: many people can benefit from one person's contribution, and the users need not do work that someone else has already done.

It is a challenge for usability to make the editing system precise enough that the editors will enter acceptable data, but easy enough that the editing does not require expertise in linguistics or ontologies. The front end of TF is implemented on Mediawiki[3], and the term editor is based on CKEditor[4]. Figure 3 shows the view for TF.

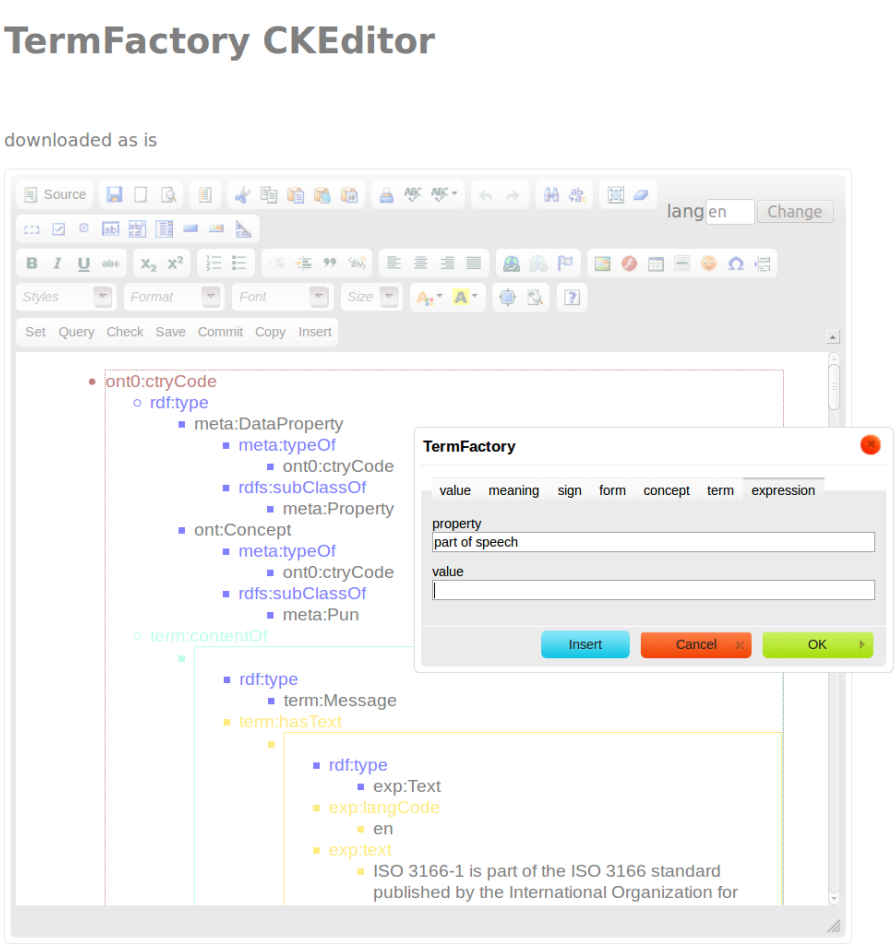
For concrete motivation, we consider short examples of the type of things that TermFactory could be used for. A possible scenario is that we have an ontology that has no name predicate, that is, the concepts are represented by URIs or other identifiers not directed to human readers. In TF we do not modify the original ontology—it need not even be physically located on TF server. Instead, we make a local extension on TF platform, enhancing the ontology with the data we need, in this case names or labels in natural language(s).

Another scenario is to map ontologies to other ontologies (section 4.4.2). Say we have ontologies of different museums, and each has information about artworks, artists, painting materials and such. They use slightly different triples, but basically all express very similar things. Instead of every mu-

[3]http://www.mediawiki.org/wiki/MediaWiki

[4]http://ckeditor.com/

Figure 3: Edit view of TermFactory



seum doing their own conversion from ontology to GF concrete syntaxes, we could build a general museum ontology in TF platform or choose one of the existing ontologies, map every museum's own ontology to that, and write only one GF grammar, for the general museum ontology.

In this project, TF will be used as a bridge between GF and FactForge. FactForge does not include morphological and syntactic information, which means that it is not possible to generate GF grammars completely automatically from the ontology. TF is used as an intermediate step; the user brings the terms over to TermFactory for the community to review and enrich the entries. The process is discussed in more detail in section 4.4.

3.4 WordNet

WordNet is a lexical database of open-class words, that is, nouns, verbs, adjectives and adverbs (Miller, 1995). It is organised as a network whose nodes are sets of synonyms, where each set expresses a concept, explained by a brief and descriptive gloss. The sets are interlinked such that from a single word form it is possible to retrieve both its synonyms and homonyms, that is, other words from the same synonym set and other senses of the same form. There are semantic relations defined between the synonym sets, such as synonymy, antonymy, hyponymy and hyperonymy, and entailment relations: for example, in order to *divorce*, one has to *marry* first.

The original WordNet project was started in the University of Princeton in the 1980's. Since then there have been updated versions of the original, and versions in different languages, either by translating the original or collecting terms from scratch (Pääkkö, 2011). The English version includes part of speech and inflectional morphological information (*go* and *went* are forms of the same lexeme), but not derivational information (*interpret* and *interpreter* are distinct entries). Other language versions might have different information: for example, the Finnish WordNet entries include the inflection types of the words, based on the classification by the Institute for the Languages of Finland[5].

WordNet is an ontology in the sense that it is a collection of concepts and relations. The hyponym–hyperonym relation is basically a subclass–superclass hierarchy, which covers the majority of nouns. WordNet has been used in many applications, such as question-answering systems and word sense disambiguation (Pääkkö, 2011). In MOLTO it could be used as a source of synonyms or glosses. In section 4.1.3 we present a method in which the lexical items in a grammar are disambiguated by explanations in parentheses; these explanations could be taken from WordNet glosses.

4 Use cases

Having presented the prerequisite information about resources and theoretical background, we move on to the objective of this study. Namely, we will

[5]http://kaino.kotus.fi/sanat/nykysuomi/

cover three different use cases: disambiguation (refsec:disambiguation), lexicon extension (4.2) and ontology verbalisation (4.3).

From the end user's point of view, the goal is to have a translation system that is useful. It has to have a comprehensive and easily extendable vocabulary, relevant to the domain of the system. Syntactic structures need not be modified on the fly—this is a considerably harder task, and not that crucial. In a typical scenario, it is not realistic to build the system to handle completely unrestricted text, at least not without a statistical back-off. When using systems based on controlled natural language, in the case of parse error, the user should change the structure of the input sentence instead of the grammar.

However, modifying the vocabulary should be possible for any user, with little effort and no technical expertise. Language skills in all different languages should not be needed either; with the ontology as a lexicon harvesting aid, one should trust to find concepts and translations to them, instead of just words, whose senses may not overlap in different languages. In section 4.2 we discuss the topic of ontology as the source of lexicon, covering both theoretical and practical questions.

Disambiguation, discussed in section 4.1, is another relevant issue in machine translation. If a sentence in the source language is ambiguous, the program could either choose one translation, in which case the result is less reliable, or show all options to user, which is a usability issue, especially if the user is unfamiliar with some or all of the target languages. Ontologies can help with disambiguation, but it may come with the cost of making the grammar too complicated and inefficient.

One option is to base the whole grammar on an ontology. In this case, it is not a question of machine translation as such, but ontology verbalisation. In terms of GF, the process consists of linearisation without the parsing phase. The ontology provides the concepts and relations, and the problem is to form a coherent and meaningful textual representation, as well as a way of conceptual authoring—the starting point is not a text written in some natural language, but a meaning, or perhaps a query for which the answer is fetched from an ontology and verbalised. This approach is discussed in section 4.3.

4.1 Disambiguation with ontologies

As mentioned in section 3.1.5, a GF grammar can include the whole hierarchy of an ontology. This kind of information is indispensable for any system that operates between text and the world. However, in machine translation, where the task is to provide a mapping from text to text, the question is reduced to finding the information that is enough to prevent erroneous translations. The practical question is how to implement it: the ontological information can be hard-coded into GF grammar, or ontologies can be used as an external source, accessed only when ambiguity is found. Another possibility is to leave the final decision to a human, but provide a *disambiguation grammar*, which gives explicit translations for words whose meaning is more broad in the source language than in the target language.

4.1.1 Integration to GF grammar

In the context of GF, an expression is ambiguous if it has more than one abstract syntax tree. The intuition of a human speaker or a reasoning system is irrelevant; an ambiguous sentence like *I made her duck* has only one abstract syntax tree in a grammar that has only one sense of the word *duck* or *her*, or only one type of complement for the verb *make*. It depends of course on the grammar in question if an expression is ambiguous, and there is no sure way of automatically solving if a grammar has ambiguous linearisations in some of its concrete syntaxes.

Homonymy in the level of atomic lexical items is not hard to detect: if any two abstract syntax lexical functions have identical linearisations in a proper subset[6] of all languages, there is a possibility of ambiguity when translating to languages in which the two concepts are not homonymous. If the two elements belong to the same abstract syntax category, that is, they could be interchanged in a sentence without modifying the nonterminal nodes in its abstract syntax tree, the potential ambiguity is of a lexical sort. If the two elements with identical linearisations belong to different abstract syntax

[6]If the linearisations of the two elements are identical in all languages in the system, none of the languages would be ambiguous in relation to others, so from an MT perspective, there is no problem.

categories, it might cause syntactic ambiguity, given the right combination of other polysemous or homonymous constituents.

Domain-specificity makes the task easier; it is really a contrived example to assume that both bass singers and bass fish appear in the same grammar in exactly same kinds of contexts. In a domain-specific system, more likely source of ambiguity is syntax. Lexical ambiguity tends towards polysemy; it is more about fine distinctions where different languages draw a line differently. Authentic examples from the case studies include the painter-subject distinction discussed in section 3.1.5, and from the same corpus, another preposition *på* can be used to express both that an artwork is painted on some base, such as canvas, or that the artwork is located in some place, such as museum—similar to English *on*: painting (hanging) on the wall or painting (painted) on canvas. A polysemous example from the domain of mathematics would be the modifier *even*: there are even distributions ('balanced') and even numbers ('divisible by two')[7], both meanings very common.

The key to this approach of disambiguation is the GF type system and its control over legal combinations; it is a syntax error to give an element of one abstract type to a function that requires an argument of some other abstract type. In case of ambiguous structures, the constituents can be marked so that only one parse is possible. As shown in section 3.1.5, we can separate painters from subjects and bases from places by giving them different abstract syntax types. Using dependent types and proof objects helps fine-tuning the grammar and separates syntactic well-formedness from semantics. The dependent types make it possible to distinguish the most minute details if needed, and they can be modified without having to refactor the rest of the grammar.

Of course this solution is still not perfect, since a painter can be also a model, and a wall could be either a location of a painting or its base. The hard-coding of the ontological information to GF has its limitations: if the context is not specified in the immediate constituent where the ambiguous term appears, the extra information provided by the type system does not help. A contrived example: *I went to the bank* could mean either the monetary institution or the river edge, only disambiguated by whether the sentence continues *and withdrew 200 euros from my account* or *and walked along the river*. In the grammar, the scope of the function that handles the argument *bank* would not include any

[7]And by polysemous extension, even polygons ('number of sides is divisible by two')

continuing clause. While it is technically possible to do ontological reasoning in GF type system (see page 26), currently that is not fast enough to be a feasible option (Angelov 2011: personal communication).

4.1.2 Ontology as an external source

Another possibility is not to integrate the ontology to GF at all, but use an external ontology and a reasoner. There has been research on using semantics as grammar rule constraints (see section 5), however, GF has not been used. As there is no practical research done, the whole idea stays very abstract. The basic idea is to have the GF grammar in a format that could be given to a reasoner that uses semantic constraints to filter infeasible interpretations. When encountering an ambiguous parse, all abstract syntax trees for the sentence are extracted and given to the engine for filtering.

If a text is parsed by a GF grammar, we already have semantic representations for the words in the abstract syntax. If this step becomes relevant, it means that some part of the sentence gives possibility of two or more parses. The previous method relies on detecting and solving ambiguities beforehand; the user could go through a lot of trouble over something that never occurs in any real life situation. This approach does not declare any combination as strictly illegal, but in a competition against a more semantically sound parse, a semantically weird interpretation would lose. If the semantically weird parse is the only parse, it depends on the priority of the user what to do; if the goal is just to prevent ambiguity, the whole process of ontology check need not be done if GF returns only one parse.

For this approach to be useful, the grammar would need to be well designed with respect to the abstract syntax types. The GF implementation would not need full information about the taxonomy of its lexicon, but the abstract syntax types need to be chosen well enough that when given to an ontology-based reasoner, the types would correspond to the items in the ontology, so that any reasoning can happen.

4.1.3 Human-aided disambiguation

The distinctions in deictic expressions, such as personal pronouns, is another issue, which is not solved by domain-specificity. Suppose the source language is English and the input sentence *I love you*. We could get various translations in different languages, showing variations according to gender and familiarity of the participants. The variation can show in the choice of pronouns, in the inflection of the verb or in both of them. Without any additional information, a monolingual English speaker would not know which translation to choose.

Instead of making the program decide, we could write one more concrete syntax, a disambiguation English, or whichever is the grammar writer's language of choice. Its lexicon would be identical to the regular English, except for the homonymous terms, which would include explanations in parentheses. In the example sentence, personal pronouns cause ambiguity when translating from English to e.g. Spanish or German. The abstract syntax of the grammar would have terms like `I_Female` and `I_Male`, `you_SgPoliteFemale`, `you_PlFamiliarMale` etc. The English linearisations for those are just *I* and *you*, but disambiguation English would state explicitly *I (female)*, *you (singular, familiar, male)*. This way, when one phrase in English gets many translations in some language, all translations would be accompanied by the disambiguation English translations.

A disambiguation grammar can be done for every language, and disambiguations can be shown in the language from which the user is translating. In different languages different terms would need disambiguation. Coming up with informative glosses can be hard, but there are resources: the glosses could be extracted from WordNet, provided that the language has one. In WordNet (see section 3.4), the words are arranged in synonym sets, and each set is given a description of its meaning. The descriptions are relatively short, but detailed enough to disambiguate meanings. An example of the results for the adjective *even*:

> The adj even has 6 senses
>
> 1. even — (divisible by two)
> 2. even, fifty-fifty — (equal in degree or extent or amount; or equally matched or balanced)

3. even — (being level or straight or regular and without variation as e.g. in shape or texture; or being in the same plane or at the same height as something else (i.e. even with))
4. even, regular — (symmetrically arranged)
5. even, regular — (occurring at fixed intervals)
6. tied, even, level — (of the score in a contest)

A human has to choose the right explanations—a mathematics grammar would need at least the first two kinds of *even*, but WordNet has six different options. Even if there were only two senses for the word, it would be still hard to automate the decision of which one is which. However, it could be still useful to have WordNet as a resource; choosing from a list of descriptions is less work than to design good descriptions from scratch. If glosses in natural language are not available or they prove too clumsy, the description can be just a more specific subclass, for example *even (arithmetics)*, *even (geometry)* and *even (statistics)*.

An important question is how much does an ontology really help. An ontology can provide information of the taxonomy of a concept, but generally only a human knows which distinctions are important in a grammar. For example, a painting base and a location could be separated from each other in many ways, all relatively easily extracted from the ontology, but most of them are not very useful. However, when designing the grammar structure, it might be helpful to model it in accordance with an ontology: all approaches on disambiguation, whether it happens within the GF grammar, by an external ontology or aided by human, assume or at least benefit from a coherent, well-structured abstract syntax that is compatible with the ontology.

4.2 Ontology as the source of lexicon

Potential ambiguity of the translatable input is not our only concern. We want to give the users a chance to modify the lexicon of a translation system, even for languages they might not know well or at all. One use case, as presented in the introduction, is to search from the ontology terms that are similar to the search key, and get a set of related concepts and their translations. Ontology can be used at any point or by anyone: a grammar writer at the development

stage, or a user who wants to extend the vocabulary of a grammar already in use.

This method of acquiring the lexicon would not require any further disambiguation, since the terms are chosen by their meanings to start with. A situation with only one *even* in a mathematics grammar could happen if the lexicon is designed with focus on words. A more systematic method would be to harvest terms on different subdomains. Even though all *evens* are already domain-specific terms, in the domain of mathematics, they can still be divided into smaller subdomains: arithmetics, geometry and statistics. If the lexicon is harvested from a mathematical ontology, with the idea to get all concepts belonging to a certain category, then one *even* enters the lexicon along with other terms in arithmetics and another when fetching all terms in statistics.

The method is best suited for finding nouns. If we think about the structure of an ontology, subjects and objects of a relation are usually nouns. Predicates include some verbs or verb phrases (e.g. `holds_account`), but often they are relational nouns or adjectives, possibly combined with verbs such as *is* or *has*, or passive verbs (e.g. `funded_by`). Even if there are some verbs, often predicates do not have a very detailed hierarchy; superclass of all properties is `Property`, with subclasses such as `Object property` or `Datatype property`. For a predicate such as `flows_through`, there most likely would be no intermediate superclass related to movement, it would be just some kind of property.

Another practical issue is how to access the information. There are ontology query languages, such as SPARQL, for making structured queries. Let us look at an example query that returns all painters whose paintings are in Louvre. First it searches all triples where `http://dbpedia.org/property/museum` is a predicate and `http://dbpedia.org/resource/Louvre` is an object. Then, for each subject found in previous stage, it finds all triples in which that item is the subject and `http://dbpedia.org/property/artist` is the predicate, and returns the object of that query. The final result consists of only artists of the paintings.

However, ontology query languages have relatively high learning curve, and we should not expect grammar writers and translators to be experts on query languages. We consider two options for accessing the ontology: a full

natural language query that gets translated into a query language, or a pre-constructed query that the user completes with keywords.

4.2.1 Natural language queries

The idea of natural language queries is that the user can write a specific query in natural language, and it is translated into an equivalent expression in an ontology query language. For example, *give me the names of all paintings whose artist is named Rembrandt* would construct a SPARQL query that matches those items who appear in triples with predicate `artist` and object is named Rembrandt, and returns only the names of the paintings. The example query is shown in program 9. Everything starting with a `?` is a variable, everything between `<>` is a URI and quoted text is a string literal, only possible as an object of a triple.

Program 9 A SPARQL query that returns all paintings by Rembrandt

```
SELECT ?paintingName
WHERE {
   ?painter <http://dbpedia.org/property/name> "Rembrandt" .
   ?painting <http://dbpedia.org/property/artist> ?painter .
   ?painting <http://dbpedia.org/property/name> ?paintingName
}
```

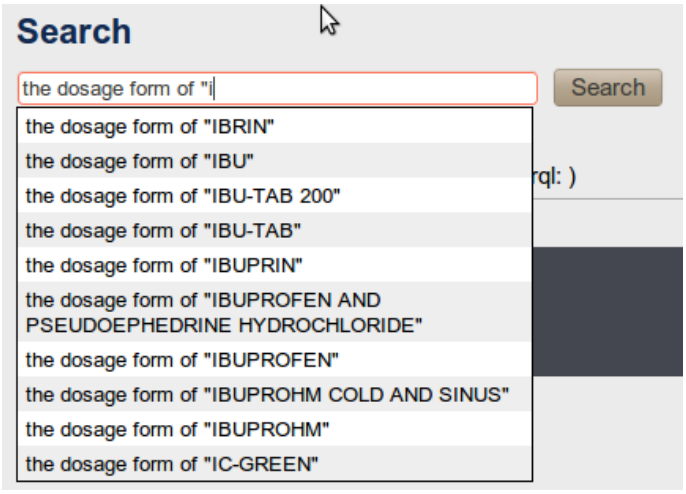
The conversion from a natural language expression to a SPARQL query is not trivial. As for technical minimum requirements, we need a mapping from natural language expressions to the items in the ontology, and a syntactic parser to figure out the relations; that is, what is their position in the triples. Same construction in natural language can correspond to many predicates in ontology, depending on the domain, and same predicate can be expressed with many natural language utterances. Our example query could be phrased *the names of all paintings painted by Rembrandt* or even *all Rembrandt paintings*, leaving the name part implicit and omitting the painter–paintee relation from Rembrandt and the paintings, just using the apposition *Rembrandt painting*.

In order for a natural language query to be feasible, it should be able to convert any type of query into query language, otherwise it would be easier just to stick to pre-constructed queries (section 4.2.2) and explain the limitations to

the user. It should also allow lots of freedom in phrasing the query; otherwise it would be just another restricted syntax, and it might be more motivating simply to learn the query language.

Previous approaches to natural language queries for ontologies include projects such as Wang et al. (2007), an all-purpose ontology NL interface that uses statistical parsing and synonym search in order to cover a wide range of expressions. Wang et al. use examples such as *which is the longest river that flows through the states neighboring Mississippi*—to parse and answer that kind of question, the system needs to connect a lot of information. Considering that these queries can be phrased in different ways, the parser has to be very robust.

Figure 4: Ontotext's natural language query for pharmaceutical patent ontology



Another direction is to limit the natural language queries to specific ontologies. Figure 4 shows Ontotext's natural language query for an ontology of pharmaceutical patents. Since the ontology contains only very specific information, it is possible to get a good coverage of potential queries. In the spirit of MOLTO, the natural language interface is done with a GF grammar that recognises multiple variants to phrase the queries. The abstract syntax trees are then transformed into SPARQL queries, and the results of the query are transformed to GF abstract syntax trees and linearised, so the output of the query is also natural language. Using a GF grammar makes it easy to create more concrete syntaxes in different languages. GF also provides incremental parsing: as the user types, the editor parses the text and suggests possible words, and prevents the user from typing non-recognised content.

There have been usability studies comparing different natural language querying systems. The results found by Kaufmann and Bernstein (2010) show that the users did best with systems that accept many variations in phrasing the questions. The most restrictive systems got bad results, both in user evaluations and in the time they spent doing the tasks.

Even if the problems with restrictiveness were worked out, either by very robust parsing or enough specialisation, the question is whether a full natural language query is suitable for term harvesting. The use cases hardly need very detailed specifications, such as the example with the longest river by Wang et al. We have much more restricted use of the ontology. Of course an option would be to develop natural language queries designed especially for term harvesting—that would allow more complex restrictions, and the results would be lists of words, not even trying to answer specific questions.

4.2.2 Keyword matching

A compromise between full natural language query and plain keyword search is to create a query template, and have the user insert relevant words into it. The core of the query consists of different `isA` and `isNamed` relations in the ontology. Entries from different sources use different predicates, but the overall structure of the ontology is coherent and predictable; the relation between salmon and (edible) fish is the same as with mononucleosis and (viral) disease. As the idea is to harvest terms and not answer questions such as what is the largest mammal in Africa, a pre-constructed query should satisfy most of the users' needs.

The technical side is very simple. If we know the name or a part of the name, we can search for a name relation in which it appears as the object. The subject of that relation is a URI that we can then use to get the term's superclass, and as a final result, search for everything belonging to that superclass. Of course this depends on the ontology's labelling system—some systems have a rich representation system for names, including morphology and syntax, and some have a simple `key` predicate with a string as an object. A former type of labelling needs a more complex query, since the names themselves are resources and not just literals. For an expert use, a thorough naming model would be desirable: it would include more structure and variation, alternatives

and synonyms, and the user could refine the search to include linguistic details. The latter kind of structure is simpler and more common; the ontologies in FactForge all use the model with a `label` or `key` predicate, sometimes with multiple options. Program 10 shows a query for this kind of labelling model.

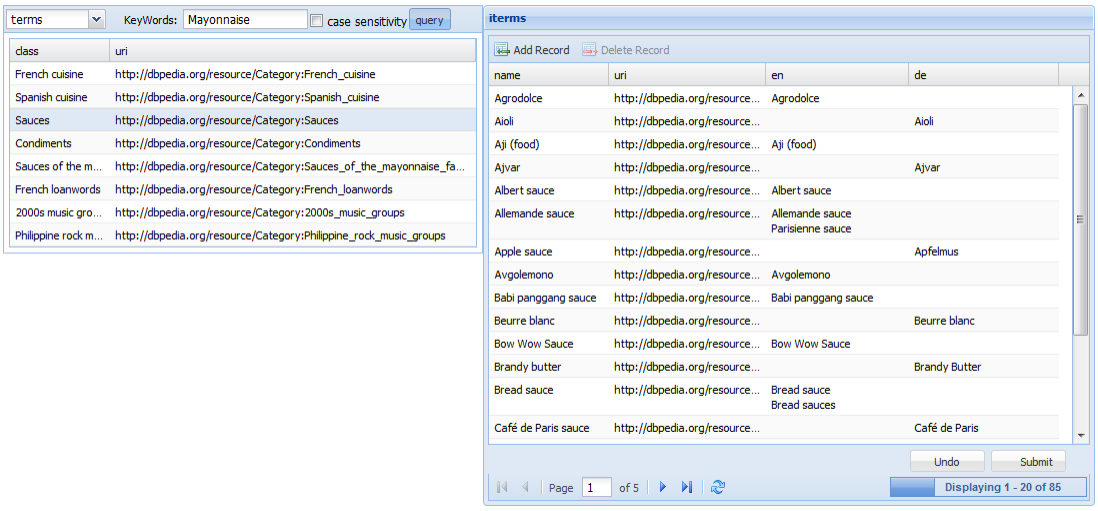
Program 10 A pre-constructed query

```
SELECT ?termName
WHERE {
   ?givenTerm <isNamed> "Name given by the user" .
   ?givenTerm <isA> ?superclass .
   ?wantedTerm <isA> ?superclass .
   ?wantedTerm <isNamed> ?termName
}
```

The workflow is such that the user types a word, which is either a superclass of the kind of concepts she wants, or a similar word. For example, if searching for edible fish, she can type *Seafood* or *Salmon*. In first case, the tool constructs a query that returns the names of the entities that have the superclass Seafood. In second case, the tool first searches for the superclasses of salmon, asking the user to specify if needed, and returns the names of the entities belonging to the same superclass. The user can specify in which languages she wants results, and if some term is not found in one of the languages, the program can substitute the term with its superclass: if there is no Catalan translation for salmon in the ontology, instead of leaving the entry blank, it is replaced by the nearest superclass that has a Catalan translation.

A user with no skills in query languages would benefit from a way of searching with just words, but for an experienced user, pre-constructing a query limits the options. For example, someone might want to search for the names of all paintings whose genre is postmodernism. This would not work in the previously described model, because an art genre is not the superclass of the painting, it is a predicate with no use outside of art domain. Foreseeing all possible things that a user would like to use to harvest vocabulary is hard, same way as it is with a full natural language query, so the translator's editor should make it possible to write SPARQL directly or from TermFactory's query form. The results, from keyword matching or native SPARQL query, can be

Figure 5: Query user interface



then added to TermFactory and converted to GF format, either with TF as a bridge or directly from FactForge.

4.2.3 Converting the results into a GF grammar

Converting the results into a GF grammar is the most practical phase in the process. The part of deciding what type of query to use is not obvious; different approaches have their good and bad sides, and there is no single best solution for all situations. However, in this part we assume that the user has done a query and got results, and the next step is to add them to a GF grammar. Ideally the harvested terms would go to TermFactory for revision and additional details, and the conversion to GF would happen from TF; this process is described in section 4.4. However, if the user wants to add terms to a GF grammar immediately, it should be possible.

In order not to break any existing file, it would be best to create a new lexicon module for each term harvesting session and import them into the main GF lexicon. We get the abstract syntax function name automatically: if the URI is `http://dbpedia.org/resource/Patagonian_toothfish`, we just get the name part after the last slash, `Patagonian_toothfish`. If there are non-ASCII characters in URIs (in which case they are actually IRIs), they should be mapped in a way that does not lose information; for instance `ä` with `a'` or `ae`, depending on the convention. The only nontrivial part is to decide the abstract syntax type of the harvested terms. One option is to ask the user, or

for a scenario where the user needs absolutely no knowledge of GF, the whole search can be connected to an existing grammar. The example used in the search must be something already in the grammar, and everything found by the search will share the example term's abstract syntax type. If the example term is not in the grammar, something returned by the query in some language might be, and the editor could check that. A worst case scenario could be just to add a placeholder type—either use an invariable dummy or choose one of the types that are in use in the grammar—and warn the user.

Generating the concrete syntax is a similar task. Assuming that the query has found a correct abstract syntax type for the new terms, for example `Fish`, they are given the same type in concrete syntax as other fish. If the abstract syntax type is not found, the concrete syntax type should also be just a placeholder. At least the structure of the file can be generated automatically, so that all material that the grammar writer needs is already in place.

To continue the example, we assume that the type is not a problem. The term *Patagonian toothfish* needs to be added in 4 languages: German, Polish, French and English.

```
de : Schwarzer Seehecht
pl : Antar patagoński
fr : Légine australe
en : Patagonian toothfish
```

The editor can generate for each of them a concrete syntax: `FishGer.gf`, `FishPol.gf`, `FishFre.gf` and `FishEng.gf`. A good concrete syntax category for nouns is CN instead of the lexical category N (Hallgren et al., 2012, p. 31), so that should be their type. Single words can be converted into GF common nouns just by writing `mkCN (mkN "word")`, but multi-word concepts are harder. For example, in Polish and French the *toothfish* is the first word, but in English and German it is the second. Good guesses like `mkCN (mkA "Patagonian") (mkN "toothfish")` do not work for all languages. A safe choice is just `mkCN (mkN ["any sequence of words"])`, and an expert in that language can later on check the grammar and do corrections when needed.

4.3 Ontology as the (source of) grammar

The distinction between an ontology-based grammar and the previous two approaches is in the degree of ontology use. When using ontology for disambiguation or as the source of lexicon, the process of grammar writing is not guided by the ontology as such. The user decides what kind of expressions are supported by the grammar, and may use an ontology to get ideas what kind of categories should be included, but ultimately the grammar is built with the idea of translating text to text.

With the ontology as the grammar, the idea is to eliminate the parsing phase, and generate sentences by verbalising the ontology. This is not machine translation, but multilingual natural language generation. The system does not need to prepare for unrestricted vocabulary, and user input, if there is any, comes in the form of a query, not as a source text to be translated. We include this approach in the study, because the methods of building this kind of system do not differ from building a translation system: the GF grammar is similar, but it is not used for parsing, only for linearisation.

4.3.1 Ontology verbalisation

As introduced in section 2.2, the information in an ontology is organised in triples, each presenting one fact, and verbalising them directly would result in clumsy sentences. If the goal is just to provide a natural language translation to browse the ontology, this might be enough. However, in order to produce fluent passages of text, the process must involve information aggregation and discourse planning.

The museum case study in MOLTO uses the latter kind of ontology verbalisation. We have the Gothenburg City Museum's ontology of their collections, and the objective is to generate automatically various types of descriptions, such as leaflets, web pages and interactive applications, where the user can search information via directed or natural language queries (Dannélls, 2011). The platform for querying or browsing could also be a semantic multilingual wiki developed in another work package of MOLTO; more on that in section 4.3.2, where we discuss user input.

Each format requires a different kind of information structure and details. The technical implementation is a GF grammar (Dannélls et al., 2012) with dependent types (section 3.1.5) to guarantee compatibility with the ontology. The grammar is structured in two parts: lexicon and text patterns. In the abstract syntax there is in addition a database of proof objects for each painting, to make sure that only existing combinations of data are linearised. The first excerpt in program 11 is from the Swedish concrete syntax of the lexicon, and it is a linearisation of a painting. The second one is the proof object which asserts the properties of the painting. The painting `GSM940042Obj` can then be used by different kinds of discourse building functions.

Program 11 Linearisation of a painting and a proof object

```
lin GSM940042Obj = mkPainting "Peter Ulrik Ekström";


data GSM940042ObjPainting : CompletePainting
  GSM940042Obj MiniaturePortrait JKFViertel (MkYear (YInt 1814))
  (MkMuseum GoteborgsCityMuseum) (MkColour Grey)
   (MkSize (SIntInt 349 776)) (MkMaterial Wood) ;
```

There are no linearisations in the concrete syntax for the database; elements such as `Grey`, `Wood` and `MiniaturePortrait` are linearised in the lexicons of each language, and the database is just for creating valid combinations. Furthermore, porting the GF grammar to a new language does not require touching the proof objects. Only if new paintings are added to the lexicon, one has to make corresponding proof objects for them.

The discourse building functions in the concrete syntax take different arguments and expose the information in different ways. For instance, the author of a painting is one of its key elements, so it would appear in nearly all discourse patterns. Details such as colour and size are less important for a general description about the painting—then again, for a direct question about the painting's size, the user would not need its author and art genre in the same response. The choice of what information to include is a question of a communicative need, but the order of the information and other grammatical aspects vary between languages: for example, in English it is natural to use a passive when describing that a painting *is painted by* an author, and in Finnish

it sounds more natural to use an active construction with an inverted word order.

Could the text generation be more general? A short pattern such as *Painting was painted by Painter in Year. It is of Size [Width by Heigth] and it is painted on Material. This PaintingType is displayed in Museum.* shares a common information structure with any description, for example *Artefact was developed by Manufacturer in Year. It measures Size [Length m and Weight kg] and it is made of Material. This ArtefactType is displayed in Museum.* From a technical point of view, it is possible to abstract away the common parts of the text generation templates as an incomplete module, and instantiate a text generation template for art museum by using the verb *paint* and for a war museum by using the verb *manufacture*.

This kind of parametrised text template goes well with the grammar design of base module and domain extensions; see e.g. Hallgren et al. (2012). A base module `Museum` would contain categories, predication functions and user-friendly constructors. Domain extensions `ArtMuseum` and `WarMuseum` would each contain a domain specific lexicon, easily extendable by an end user. As a minor technical annoyance, the module structure would become more complex—in the current model, in order to create a new language in the art museum ontology, only two real modules and a three-line dummy wrapper need to be written for each. The benefit in the parametrised template model is that once someone has written a generic template for descriptions, one can use that for different types of museums only by changing lexicon (*paint/manufacture*) and some domain-specific details such as size units: a painting is described by two dimensions, whereas a tank needs three dimensions and a weight.

However, it is important to consider whether different domains are really compatible, especially when moving away from toy examples. We have attributes like material and colour; for a painting it is natural to mention them even in the first sentence, e.g. *Mona Lisa was painted on canvas by Leonardo in 1503*, but a description such as *Leopard 2 was produced using metal and plastic by Krauss-Maffei in 1979* does not sound natural. Especially for longer texts with more details, it would be difficult to create interchangeable discourse patterns. A more realistic use case would be items within the same domain that have only minor lexical differences. For example, paintings and sculptures would have enough in common to use the same discourse patterns, but differ-

ent word choices, like *painter* and *sculptor*. Dannélls (2011, p. 5–6) refers to Chenhall and Vance (2010), who find that the core metadata is similar in museum records from various domains, and implements the design of the painting ontology accordingly. Nevertheless, it is probable that these basic attributes, such as object name, origin, value, creator and location, would not even need different verbalisation parameters in different domains. Thus it remains to be seen how much a need there is for a solution like this.

We return to ontology verbalisation in section 4.4.2. Rather than different domains using the same generic text generation template, we will discuss different ontologies of the same domain using the same verbalisation grammar with the help of TermFactory, using ontology alignment.

4.3.2 User input

The question of the user input in the museum case has not been discussed very much. The generation of leaflets and web pages is straightforward; the responsibility is with the grammar writer, who decides what type of information is crucial, and then the chosen discourse patterns are applied to all paintings. This is a typical MOLTO workflow, as defined in 2.3: the initial effort is to decide the information structure, and the text is generated automatically in many languages. The grammar is not different from any other grammar, only the units in the abstract syntax are complete paragraphs instead of single sentences that can be combined in any way. If a new piece is added to the collection, a similar text about it can be generated instantly. This is analogous to the case where a website is updated in one language and the update is simultaneously translated into other languages. In this scenario the source is not written in a natural language, instead the text comes from applying the chosen pattern to an ontology record.

The planned interactive applications are not that straightforward. In the following paragraphs we will see three methods of querying the ontology: free natural language query, guided natural language query and a semantic wiki.

Free natural language query In section 4.2.1 we discussed ontology queries as a means of acquiring lexicon; this is a similar problem, but with different ends. The user would type a query about museum objects in natural language

and get a response in natural language, built with the discourse patterns presented in 4.3.1. This means two grammars: one for the text patterns and another for the query. However, the query grammar would most likely be easier to make than the discourse pattern grammar or any generic query grammar. The domain is limited and the vocabulary is restricted; the problem of one-to-many (*painting painted by Rembrandt* and *Rembrandt painting*) still exists, but at least the apposition *Rembrandt painting* would not refer to a movie directed by Rembrandt, a book written by Rembrandt or a course taught by Rembrandt—that is, we can expect to get rid of most cases of many-to-many relations. There are still many ways to say that someone is the author of a painting, but with a restricted domain, we can expect that each of those natural language expressions refers to only one relation in the ontology. Otherwise, the pros and cons of a natural language query have been discussed in 4.2.1.

Directed natural language query Like the previous method, the result of this one will be a natural language query, but the process of formulating the query is guided. The guidance can be provided at a semantic or syntactic level. A technique called *conceptual authoring* or *WYSIWYM*, "What You See Is What You Meant", is an example of the former. Figure 8 shows an example of a conceptual authoring UI to a tourist phrasebook, written by Michal Boleslav Měchura.

The image is the first view of the phrasebook. The phrases are grouped by meaning, and one phrase can belong to more than one class: expressing that someone wants to go somewhere is under two titles, wants and places. After choosing one of the broad scenarios, the user needs to fill the details: for example, someone wanting something requires information of who wants and what. Besides semantic valency, there are grammatical details, such as degree of politeness, number, gender and definiteness of the arguments and whether the action is a statement or a question.

The generation of a WYSIWYM interface is not a trivial task; however, the grammar itself could be used to semi-automate the construction of the interface. The action categories in the WYSIWYM Phrasebook all come from the functions in the abstract syntax: `AWant : Person -> Object -> Action` becomes *someone wants something*. A grammar used with a WYSIWYM interface would need a generic indefinite member of all participants—*someone*

Figure 6: Tourist phrasebook: WYSIWYM

for `Person`, *something* for `Object`, *somewhere* for `Place`—which would be used when generating a template of an action. The platform, for instance a web page or a mobile application, would be reusable with any grammar. In addition, a start category for a top level action in WYSIWYM version would be specified in the grammar; in Phrasebook it is `Action`. Then the WYSIWYM editor would create generic sentences out of all `Action`-producing functions and use them as models in the first view. Human effort is required in the grouping and naming of the example sentences; for example, deciding that *someone loves someone* belongs to the category *Relationships*. To add multilinguality, the platform needs to be translated for each language, but the model sentences for a given grammar can be generated for any concrete syntax of that grammar.

Hallett et al. (2007) have tested a conceptual authoring tool for composing questions for a database of medical histories. The subjects succeeded in composing fairly complex queries, such as *How many patients between 30 and 70 years of age, who had a clinical diagnosis of malignant neoplasm of breast and underwent surgery, had a haematoma after surgery?* with the help of the tool. As a result, the program gets an unambiguous representation without

the need for a parsing and semantic interpretation, and the user can still see the query in natural language, so that it is easy to check whether it matches with the intended meaning. The forming of the question is guided by the program, and the result is a controlled language, but the user does not have the responsibility to learn the rules of the language.

Figure 7: Tourist phrasebook: fridge magnets

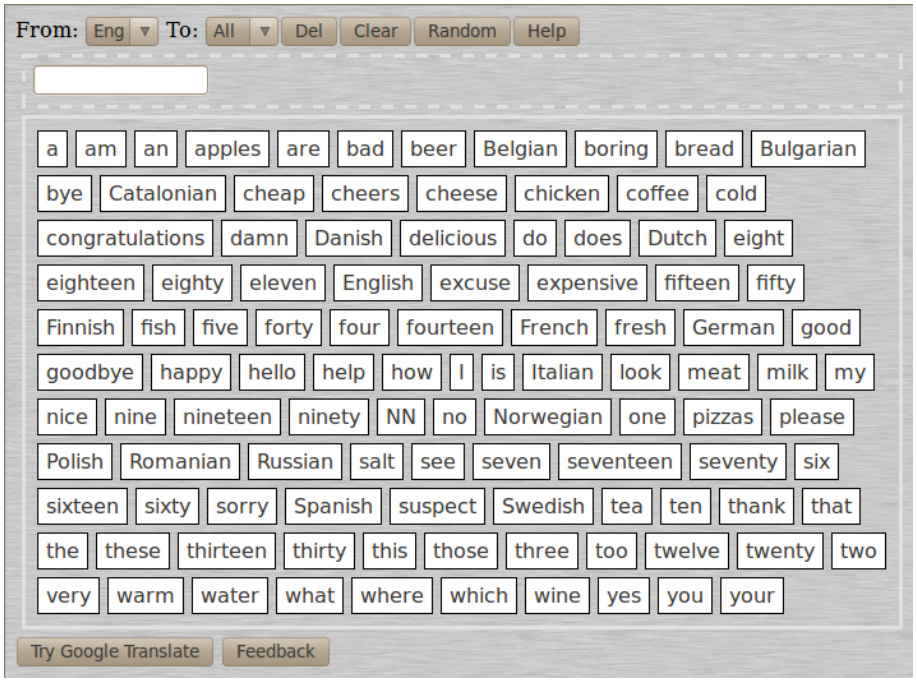


Figure 7 shows an example of directing a sentence at a syntactic level, in the style of fridge magnets. For the same phrasebook application, there are 107 possible first words for a sentence in English, as opposed to 8 possible action types. For a language that has more inflection, the number rises quickly: the Bulgarian version has 369 possible first words, because the grammar allows variation in grammatical details. In the WYSIWYM approach, those details are defined only after the basic meaning has been chosen.

Using a fridge magnet user interface is reasonable for small applications, where the languageis very restricted, and the fridge magnets are an option to completely free typing. A slightly modified version of the fridge magnets would classify the words such that the user would be shown only content words and start building the sentence from the middle. For instance, choosing *cost* or *toilet* for the keyword would limit the options to the kinds of sentences where those words can appear; the first one would be demanding or negotiating a price

of something and the second one probably asking for a location of a bathroom. In practice, the solution is not applicable: fridge magnets get potential next words by using the GF parser, which cannot start from the middle of a sentence (Angelov, 2009).

A predictive editor is another option to guide the user at the syntactic level: when the user starts typing, the editor shows possible completions and gives feedback instantly, if the word or expression is not recognised by the grammar. Conceptual authoring is an option to consider for a system that is too small to support free text, but large enough that the fridge magnets or predictive editors are impractical.

Semantic wiki AceWiki, presented in Kuhn (2008), is a semantic wiki that works on Attempto Controlled English (ACE). An introduction to ACE can be found in Kuhn (2010, ch. 2.2). The plan in MOLTO is to make AceWiki support multilingual GF grammars; see Camilleri et al. (2012).

The semantic wiki contains the statements from the ontology as natural language sentences. In addition to explicit statements, AceWiki does inference, such that statements *every painter is an artist* and *Claude Monet is a painter* result to Monet being an artist. ACE statements can be more complex than one ontology triple; for example *Mona Lisa is an oil painting that is painted by Leonardo da Vinci and that is located in the Louvre*. This aggregated statement is broken down to simple statements by the ACE parser (Kuhn, 2010), so information can be added to an ontology, even though it is not directly translatable to RDF. A common use case, anticipated by the developers of the wiki (Kuhn, 2010) is to make users add statements to the wiki, and all new information that is well-formed and consistent with the old is added to the ontology.

In this scenario, the museum ontology would have its own AceWiki, containing statements that are verbalised from the ontology. The previously introduced GF grammar for museum texts can not be used by the wiki, because everything on AceWiki has to be written in ACE. However, the ACE grammar library (Camilleri et al., 2012) supports AceWiki subset of ACE in 10 languages; all relevant ACE constructions are covered, and the extension to museum domain is purely lexical.

The wiki approach makes discourse planning unnecessary; an article is just an aggregation of the information stated and inferred on the topic. Information can be given priorities based on common tendencies, for example, a collection of statements about a painting would include its author in the first sentences and its colour as secondary information.

Figure 8: Information representation in AceWiki

In effect, a semantic wiki is a way to browse an ontology—and extend it by using controlled natural language that is readable by a reasoner. If there are inconsistencies, they are shown as conflicting and not added to the underlying ontology. As for querying, the AceWiki supports questions, such as in the image "What is painted by Leonardo da Vinci?" and when new information is added anywhere on the wiki, the answers are updated instantly to the page where the question is written.

GF will be used to make the wiki multilingual. The scenario is back to the original: a GF grammar is used to translate statements that a human writes manually or semi-automatically based on the ontology. Furthermore, the statements are translated to ACE, which is a formal language. The users

can actually extend an ontology by writing controlled natural language, multilingually.

4.4 Role of TermFactory

In section 4.2.3 we saw a quick and ad hoc way to transform the information from FactForge to GF grammars. A preferable option would be to use TermFactory as a bridge between the two, or later, even as the primary source for terms. The scenario in 4.2.3 assumes an ontology with a primitive labelling system—if the term *Patagonian toothfish* were already structured data, with information that *toothfish* is the head of the noun phrase, we could just import the terms to GF directly from the ontology. The philosophy of TermFactory and similar approaches (such as Cimiano et al. (2011), presented in section 5) is to make the step from ontology to natural language smaller.

It is good to preserve a simple `key` predicate with a string object; preferably with lot of options, so that finding the term with a keyword search would still be possible. In addition, the terms should be linked to a linguistically detailed representation without replacing the keyword relation. This can be done on the TF platform.

4.4.1 Linking ontology entries to morphological resources

In the MOLTO scenario, the terms are harvested from FactForge, or from any source, and carried to TermFactory, where a community of users revises them and adds all kinds of linguistic details needed in NLP applications. In a language with rich morphology, the information found in FactForge is not enough. Adding the details immediately after retrieving is a lot of work for one person, and if everyone does it that way, it is probable that people end up doing work that someone else has already done at some point. In addition, a term search does not provide a lot of help for filling in the information; it is specialised knowledge to know what is for instance a conditional or a gerund. As said in 4.2.3, it is not trivial even for a linguist to know what information a GF resource grammar constructor requires in order to build words.

The sensible first step is to make use of all existing resources: for example, synonym lists, frequency lists and morphologically annotated word lists that

are freely available for several languages. Before asking the user to provide morphological information, the word should be searched from an existing morphological lexicon. This solution requires mapping the classification system used by the lexicon to GF morphology; in practice, that should not be too hard, if the resource grammar morphology is made in a sensible way. A conversion for Finnish word list[8] (for Finnish Languages) has been already done by Aarne Ranta. The work is to map the inflection types of the list, represented by a number from 1-78 and an optional letter from A-M, to the functions in the Finnish resource grammar, represented by example words. Program 12 shows an excerpt from the conversion grammar.

Program 12 Mapping the inflection types used by the Finnish word list to GF smart paradigms

```
oper

  d06 : Str -> NForms
    = \s -> dTohtori s ;
  d07 : Str -> NForms
    = \s -> dArpi s (init s + "en") ;
  d07A : Str -> NForms
    = \s -> dArpi s (init (weakGrade s) + "en") ;
```

However, there might be no resources available for some language, and the platform should be prepared for words that are not covered by any of the existing resources. For that reason, the TermFactory entry editor has an input template that lets the user give the information in an easy way, providing examples. GF could be used as a back-end for the morphology. The GF resource grammar constructors take in general from 1 to 5 forms of a word to form a complete paradigm; worst cases are around 10 and very rare.

A user interface would ask the user to write a basic form, typically nominative or infinitive, possibly using an example and an advice to write the entry word in the same form. The underlying GF web service would then run a `mkN`, `mkV` or similar and generate the whole paradigm. For highly inflecting languages it would show only some key forms. The user would correct what needs to be corrected and then accept the paradigm. In case there is

[8] http://kaino.kotus.fi/sanat/nykysuomi/

nothing to correct, the way to construct that word in GF would be just `mkC <word>`. The smart paradigms in GF constructors have an order of what forms to include; for instance, a Finnish noun constructor with two forms takes the singular nominative and the plural partitive, so those would be the first forms to show to the user. When that is accepted, with or without correction, the user could be shown the next form, singular genitive. After 4 forms there is only the worst case constructor, 10 forms, and if all are accepted, then the paradigm of the word is saved to the entry.

TermFactory is meant to be compatible with all kinds of NLP applications, not just GF. It is relatively easy to get any kinds of representation formats from the full paradigm; especially if there is a mapping from some existing morphological resource to GF, it could be at least partially two-way. The GF format would be the `mkC` constructor taking the forms that the user wrote or corrected. Even if this kind of minimal constructions were not created, the words could be built with the worst-case constructors, which always have fixed forms.

Ideally, TermFactory would grow and include terms from many sources, so that the user could search terms straight from TermFactory. The search might be implemented as a single query that searches FactForge and TermFactory at the same time, and if some or all of the terms found in FactForge already have a TermFactory entry, the query would return that information as well. The pre-constructed query is currently made to work in FactForge, but in future it could be updated to cover also terms that have come to TermFactory elsewhere and not interlinked in FactForce collection. The native SPARQL query would of course work for any data.

4.4.2 Ontology alignment

Ontologies can be mapped to other ontologies. If one ontology has a mapping to some NLP resource, it is easier to map other ontologies to it than write such a resource to all of them. For instance, SUMO has a mapping to WordNet; linking any other ontology to SUMO gives that ontology an access to a network of natural language concepts.

The two ontologies can be merged into one, such that the contents of ontology `A` are rewritten in ontology `B`'s terms. This can be done manually or

(semi-)automatically. This option is good if the development of `A` is not continued; having only one ontology takes less space and is simpler than bridging two ontologies.

If the ontologies need to be kept as distinct, they cannot be merged. Instead, the contents of `A` can be mapped to `B` with `sameAs` predicates. This is the most practical solution if the ontologies are distinct entities, so that `B`'s developers need not care or even know about `A`. TermFactory supports having third-party ontologies with read-only access; the links from the user's own ontology to the third-party ontology are defined on TF, visible in the scope that the user wishes.

As a technical issue, two ontologies are not always easily mapped. A property in one ontology might best map to a union of two or more properties in another ontology, or it might have no equivalent at all. Some correspondencies depend on application, and require a human to decide. If a painting ontology has a verbalisation grammar and a music ontology is mapped to it, probably the verbalisation is not suitable for musical pieces in all aspects. For example, a song does not have a physical size, but it has a duration (a recording) or an estimated duration (a score)—should that be mapped to a size attribute? In 4.3.1, we discussed a parametrised verbalisation grammar, for domains with similar information structure but different lexical choices. This could provide a solution, but it has not been tested in more than a toy scale. However, if it turns out to be useful in some applications, TF and GF provide all that is needed to implement such a system.

4.5 Testing and evaluation

Currently we have a prototype of the search tool that works with the ontology, and connecting the ontology search to the editor is the next step. TermFactory has only recently (June 2012) been opened, and it does not yet have a user base needed for it to function as planned. The whole package is not at a point where it could be reasonably tested, but meanwhile we can test the individual components and plan how to test the first prototypes of the complete product. Thus this section is not about concrete evaluation, but sketching of ideas that should be considered when we have something more stable.

We can test the search tool independently. Precision and recall, the basic measures for any kind of information retrieval, can be used with the term search. Precision is the ratio of found relevant instances to all found instances, and recall is the ratio of found relevant instances to all relevant instances in the ontology. Precision can be determined by looking at the results; if the user has searched for edible mushrooms, the results should not contain anything poisonous. Recall needs a gold standard; that can be for example a SPARQL query performed by an ontology expert. The search results can also be compared to those from other terminology sources, such as Nomen[9] (Wollmersdorfer, 2006), a multilingual dictionary specialised in animals and plants.

If precision turns out low, it could mean that the ontology itself contains errors, such as classifying a dog as a feline. Even without clear errors, the structure of the ontology might just be contrary to the human intuition. For example, in a same class *Portrait art* there might be names of specific art pieces like *Mona Lisa* and generic terms, such as *self-portrait* or *tronie*, when the user would want only the other category.

If recall is bad, it means that there are good terms in the ontology but the search does not find them. We assume that the gold standard is done by an ontology expert in SPARQL, and the test is done with a pre-constructed query described in section 4.2.2; probably then the pre-constructed query is ignoring some important relations. There might be some generic `isA` or `isNamed` predicate not used by the query template, and it could be just added to the pre-constructed query. An opposite problem to the one with precision occurs if a category is too narrow for the human intuition, and the wanted result would combine two or more categories. A more complicated problem is if the user wants to query something that is not covered by an `isA` relation; like the art genre example in 4.2.2, and the reason that the gold standard query succeeds is because it has searched more specific relations. This should be a separate matter to test; how often are the users' needs really covered by such a simple query template? If the tests are just pre-determined tasks, such as finding all carnivores and all paintings by Rembrandt, we might miss an important issue.

Coverage of languages is another feature to test. Out of all relevant terms, how many of them have translations in other languages than English? The

[9]http://nomen.at/

premise of the study is that ontologies are better than dictionaries for working with languages that one does not know, because they are grouped by concepts, and therefore one can trust the translation to be of the exact sense of the English word. If there are no translations for many of the terms, the ontology is a lot less useful. The query would still return lots of terms, and everything about the design of a grammar based on concepts still holds, but the grammar writer would have to resort to dictionaries. A concrete test would be to give a control group a traditional lexicon, without any structure, and compare the success in term acquisition.

In addition to the translations' availability, we need to take into account their quality. Are the translations correct and recognisable, that is, really English or Swedish or Polish, or are they scientific names in Latin? An ontology is not necessarily designed thinking about human users who want to use its vocabulary, but experts who need that sort of terms. As a final, very practical detail, the search should be robust: do we get results for "cat", "Cat" and "Cats", and if not, do we at least get informative error messages?

Testing the search connected to a grammar is more of a usability test. Some things are purely technical: the user should be able to add, remove and edit the terms found by the ontology search in an easy and intuitive way. With the import to GF option, it should be possible to store terms from multiple queries and put them all into the same GF file. One of the more complicated questions is to estimate how well does a user with no knowledge of GF handle the task? In section 4.2.3 we talked about the problem of determining an abstract syntax type for the new terms and suggested two solutions: ask the user or connect the search to the grammar such that the type of the retrieved terms is one of the types in the grammar.

Besides the abstract syntax, the automatic creation of the concrete syntax should be evaluated. Assuming that the ontology search (or later with TermFactory connection) does provide good terms in many languages, it is still not given that a casual user knows how to complete the information in the GF grammar. The GF resource grammar API requires a few forms of the words that are not predictable, and a non-expert—even a native speaker of Finnish—hardly knows that for certain Finnish nouns ending in *e* only the singular nominative is needed, and for the irregular paradigm the constructor needs singular nominative and plural partitive. However, if everything goes as

planned with TermFactory, the linguistic information could be stored there, and its conversion to GF inflection could be automated.

A test that would put everything together is to get a grammar that is in progress, use the ontology search to add its coverage and evaluate how helpful the ontology component is for the user. The test could be performed with different subjects: experts in GF, experts in the domain, professional translators and a group without special knowledge in any of the fields. The state of the grammar can also be varied: the tool's usefulness for a grammar writer can be tested with a very rough draft of a grammar, and the scenario with the translator can be tested with an already working grammar, which is to be extended. The whole process of translation, with plugins to possible translation platforms is outside the scope of this study.

5 Related work

In this section we see a couple of studies related to different aspects of ontologies and lexicon management: retrieval of terms and relations, ontology verbalisation and semantics as grammar rule constraints.

Term retrieval Many approaches on term retrieval from free text rely on the *distributional hypothesis* (Sahlgren, 2008): the words are characterised by the environments where they appear, and two words appearing in similar contexts suggests that they are also semantically similar.[10] The methods can be used to harvest terms, or also relations between them, ultimately building an ontology automatically.

Lin (1998) uses dependency parsing to cluster similar words. Instead of n-grams or windows of nearby words, the corpus is first parsed to dependency triples, and word similarity is determined by appearing in similar triples. For evaluation, the groupings produced by the algorithm were compared to Roget Thesaurus and WordNet, and they proved to be closer to WordNet than Roget is.

[10]Although distributional method may have problems distinguishing between synonyms and antonyms, since they are same type of things, e.g. young and old are properties of age, black and white are colours.

Yangarber et al. (2002) have studied the learning of generalised names in the text. The algorithm is trained with a domain-specific corpus, using seeds and creating patterns of words around the seed; patterns are learned from the seed terms, and new terms are learned from the patterns. Finding multiple categories simultaneously and providing negative categories improves the precision by reducing overgeneration.

Ponzetto and Strube (2007) create a taxonomy from Wikipedia's categories. The categorisation system of Wikipedia is already structured information, but it is not a complete taxonomy; there are missing links and redundancies, multiple inheritance in a subcategory, such as *Buildings and infrastructures in Japan*, and many types of relations (both `isA` and other) between a category and its supercategories. They use various methods: syntactic parsing to match the heads of the categories (`British computer scientists` is a subclass of `Computer scientists`), lexicosyntactic patterns extracted from a large corpus and other, more specific heuristics. The coverage and quality are evaluated by comparing the results to ResearchCyc[11] and WordNet, and the results are competitive with the manually developed ontologies.

Paukkeri et al. (2010) also use Wikipedia to learn concept hierarchies automatically, but instead of Wikipedia's category system, they use articles. They use three methods to determine important keywords from an article. As a baseline they use *tf-idf*, which is the ratio of a term's frequency in a certain document vs. in a general corpus. Other methods are statistical keyphrase extraction (Paukkeri and Honkela, 2010) and fuzzy heuristics, such as emphasis, frequency and appearance in the beginning (introduction) and the end (conclusion) of an article.

Seppo Nyrkkö's PhD study (forthcoming) combines ontologies and free text. We find how the relations in the ontology are expressed in free text, and by using different types of evidence, we can get new relations and new items to the ontology. The evidence includes similarities in appearance and behaviour, for example being synonyms in a dictionary, having a common syntactic dependency or a common property in an ontology (Nyrkkö, 2011). This method can be used to measure similarities between terms, both in ontologies and in free text. The terms could be inside one language, for example medical terminology and layman terms—how does a term like *flu* coincide in the two

[11] http://research.cyc.com/

domains? The method can also be used between languages, to find out the words' coverage in semantic space and their overlap.

Ontology verbalisation Cimiano et al. (2011) have designed LexInfo, a project similar to TermFactory. LexInfo builds on the Lexicon Model for Ontologies (*lemon*), which links the ontology to linguistic information, including morphological composition, syntactic dependencies and semantics. Davis et al. (2012) present a joint project with GF and *lemon*, where GF is used to provide multilingual support for ontology verbalisation.

Semantics as grammar constraints Muresan (2010) uses a grammar formalism called Lexicalized Well-Founded Grammar (Muresan, 2008) to map the text to a meaning representation. The formalism has two types of ontology-based constraints: the first is semantic composition, that is, the meaning of an expression is composed of its parts. When parsing a noun phrase, an ontology is queried for validation of the semantics; for example, a noun phrase *blue shirt* is valid, because blue is a colour, and a colour is an acceptable property of a shirt. The other constraint is semantic interpretation, which is done after the semantic composition check; it eliminates the semantically impossible parse from a sentence like *I saw the man with the blue shirt*. Cimiano and Reyle (2003) translate the sentences into Logical Description Grammar and use an ontology to rule out impossible parses, in a similar manner. By parsing the text to a logical framework and identifying the concepts in an ontology, it is possible to infer whether some actions are feasible. A GF grammar does this by design, with its type system.

6 Conclusion

Throughout this thesis, we have explored one approach on rule-based machine translation, that is, mapping utterances between multiple languages via semantic interlingua. The goal of MOLTO is not a translation system to compete on the free text market, but a set of tools to build reliable domain-specific grammars, aimed for providers of information. In this thesis we have gone through different use cases where ontologies could be exploited: as a help to resolve ambiguity in parsing, as a source of lexicon and as the source of the grammar

itself. Evaluation is out of the scope of this study, but we have sketched possible features to test: coverage, precision and recall, usability and utility in different phases of grammar writing.

As for disambiguation, the considered options are to model a hierarchical ontology structure in the grammar's type system, to use an external ontology along with the grammar and to build an additional grammar with explicit distinctions, for the aid of human disambiguation. Out of these options, the first is in danger to turn out heavy and infeasible, and the second has no guarantee that the items in the grammar correspond to the ones in the ontology. The third option has already been used in small scale; the question is whether it is feasible in larger applications and if an ontology could help in the process.

Term harvesting from an ontology or other structured term collection is, in principle, an easy task. The questions are limited to practical issues, such as the usability of different retrieval methods, and how to turn a list of concepts with no linguistic detail into a lexicon that is suited for an NLP application. We concluded that a pre-constructed query where the user inserts words is most efficient solution for term harvesting, with a possibility to do a native SPARQL query whenever the pre-constructed query is too limited.

Multilingual ontology verbalisation is a promising use case for grammar-based methods. Their biggest weakness is limited coverage, which is a major issue when preparing for user input. When the goal is to verbalise predetermined content, the human effort can be used in making the output better instead of broader. Possible platforms for presenting the information are natural language queries with natural language answers, and semantic wikis. The latter option is also a way for users to extend the ontology by writing new statements—the language recognised by the grammar is a translation of the underlying ontology, so new sentences written in that language could be added to the ontology.

References

Krasimir Angelov. Incremental parsing with parallel multiple context-free grammars. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 69–76,

Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1609067.1609074.

Krasimir Angelov and Ramona Enache. Typeful ontologies with direct multilingual verbalization. In *Controlled Natural Languages Workshop (CNL 2010)*, Marettimo, Italy, 2011.

D.J. Arnold, Lorna Balkan, Siety Meijer, R.Lee Humphreys, and Louisa Sadler. *Machine Translation: an Introductory Guide*. Blackwells-NCC, London, 1993. URL http://www.essex.ac.uk/linguistics/clmt/MTbook/.

Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax, 2005. URL http://tools.ietf.org/html/rfc3986. Visited November 2012.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data—the story so far. *International Journal on Semantic Web and Information Systems*, 5: 1–22, 2009.

Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini. *Ontology Learning from Text: Methods, Evaluation And Applications*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2005. URL http://books.google.fi/books?id=gikMilZMFMMC.

John J. Camilleri, Norbert E. Fuchs, and Kaarel Kaljurand. ACE grammar library. Technical Report D 11.1, Chalmers University of Technology, June 2012.

Robert Chenhall and David Vance. *The world of (almost) unique objects*, pages 39–47. Routledge, 2010.

Philipp Cimiano and Uwe Reyle. Ontology-based semantic construction, underspecification and disambiguation. In *Proceedings of the Prospects and Advances in the Syntax-Semantic Interface Workshop*, pages 33–38, 2003.

Philipp Cimiano, Paul Buitelaar, John McCrae, and Michael Sintek. LexInfo: A declarative model for the lexicon-ontology interface. *Web Semantics*, 9(1): 29–51, 2011. URL http://dx.doi.org/10.1016/j.websem.2010.11.001.

Mariana Damova. Data models and alignments, May 2012. URL http://www.molto-project.eu/sites/default/files/D4.2.pdf.

Dana Dannélls. Ontology and corpus study of the cultural heritage domain. Technical report, University of Gothenburg, 2011.

Dana Dannélls, Aarne Ranta, and Ramona Enache. Multilingual grammar for museum object descriptions, March 2012. URL `http://www.molto-project.eu/sites/default/files/WP8-D2_3.pdf`.

Brian Davis, Ramona Enache, Jeroen van Grondelle, and Laurette Pretorius. Multilingual Verbalisation of Modular Ontologies using GF and lemon. In *CNL 2012*, volume LNCS. Springer, Springer, 2012. URL `http://www.molto-project.eu/node/1669`.

Ramona Enache. Reasoning and Language Generation in the SUMO Ontology. Master's thesis, Chalmers University of Technology, 2010.

Cristina España. SMT within MOLTO's hybrid translation system. Presentation, 2011.

Institute for Finnish Languages. Nykysuomen sanalista, 2012. URL `http://kaino.kotus.fi/sanat/nykysuomi/`. Visited November 2012.

Catalina Hallett, Richard Power, and Donia Scott. Composing questions through conceptual authoring. *Computational Linguistics*, 33:105–133, 2007.

Thomas Hallgren, Aarne Ranta, John Camilleri, Grégoire Détrez, and Ramona Enache. *Grammar Tools and Best Practices*. Göteborg, June 2012.

John Hutchins. *Machine translation: history of research and use*, pages 375–383. Oxford: Elsevier, 2006.

Esther Kaufmann and Abraham Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8 (4), November 2010.

Tobias Kuhn. AceWiki: A Natural and Expressive Semantic Wiki. In *Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*. CEUR Workshop Proceedings, 2008.

Tobias Kuhn. *Controlled English for Knowledge Representation*. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, 2010.

Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics - Volume 2*, COLING '98, pages 768–774, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. URL `http://dx.doi.org/10.3115/980432.980696`.

Adam Lopez. Statistical machine translation. *ACM Computing Surveys*, 40 (3), 2008.

Mathias Madsen. The Limits of Machine Translation. Master's thesis, University of Copenhagen, 2009.

George A. Miller. WordNet: a lexical database for English. *Commun. ACM*, 38:39–41, November 1995.

Smaranda Muresan. Learning to map text to graph-based meaning representations via grammar induction. In *Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing*, TextGraphs-3, pages 9–16, 2008.

Smaranda Muresan. Ontology-based semantic interpretation as grammar rule constraints. In *CICLing*, Lecture Notes in Computer Science, 2010.

Seppo Nyrkkö. Estimating term similarity and coverage - a statistical journey with syntactic evidence. Presentation, September 2011. URL `http://www.molto-project.eu/node/1334`.

Tim O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. *Communications Strategies*, 1:17–37, 2007.

Paula Pääkkö. FinnWordNet ja relaatioiden etsiminen korpuksista. Master's thesis, University of Helsinki, 2011.

Mari-Sanna Paukkeri and Timo Honkela. Likey: Unsupervised language-independent keyphrase extraction. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 162–165, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1859664.1859698`.

Mari-Sanna Paukkeri, Alberto Pérez García-Plaza, Sini Pessala, and Timo Honkela. Learning taxonomic relations from a set of text documents. In *IMCSIT*, pages 105–112, 2010.

Simone Paolo Ponzetto and Michael Strube. Deriving a large scale taxonomy from Wikipedia. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, pages 1440–1445. AAAI Press, 2007. URL `http://dl.acm.org/citation.cfm?id=1619797.1619876`.

Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011a.

Aarne Ranta. MOLTO Overview - review presentation 2011. Presentation, 2011b.

Magnus Sahlgren. The distributional hypothesis. *Rivista di Linguistica (Italian Journal of Linguistics)*, 20:33–53, 2008.

Jonathan Slocum. Machine translation: its history, current status, and future prospects. In *COLING*, pages 546–561, 1984.

Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu. PANTO – a portable natural language interface to ontologies. In *IN: 4TH ESWC, INNSBRUCK*, pages 473–487. Springer-Verlag, 2007.

Toon Witkam. History and Heritage of the DLT (Distributed Language Translation) project, 2006. URL `http://www.mt-archive.info/Witkam-2006.pdf`. Visited November 2012.

Helmut Wollmersdorfer. Nomen.at - dictionary of common names, 2006. URL `http://www.nomen.at`.

Roman Yangarber, Winston Lin, and Ralph Grishman. Unsupervised learning of generalized names. In *COLING*, 2002.