Multilingual Verbalisation of Modular Ontologies using GF and *lemon*

Brian Davis[1], Ramona Enache[2], Jeroen van Grondelle[3], and Laurette Pretorius[4]

[1] DERI/NUIG, Ireland, `brian.davis@deri.org`
[2] Chalmers University, Sweden, `ramona.enache@chalmers.se`
[3] Be Informed, The Netherlands, `j.vangrondelle@beinformed.com`
[4] University of South Africa, South Africa, `pretol@unisa.ac.za`

Abstract. This paper presents an approach to multilingual ontology verbalisation of controlled language based on the Grammatical Framework (GF) and the *lemon* model. It addresses specific challenges that arise when classes are used to create a consensus-based conceptual framework, in which many parties individually contribute instances. The approach is presented alongside a concrete case, in which ontologies are used to capture business processes by linguistically untrained stakeholders across business disciplines. GF is used to create multilingual grammars that enable transparent multilingual verbalisation. Capturing the instance labels in *lemon* lexicons reduces the need for GF engineering to the class level: The *lemon* lexicons with the labels of the instances are converted into GF grammars based on a mapping described in this paper. The grammars are modularised in accordance with the ontology modularisation and can deal with the different styles of label choosing that occur in practice.

Keywords: Controlled Natural Languages, Ontology Verbalisation, Multilingualism, Ontology Lexicalisation, Natural Language Generation

1 Introduction

As the adoption of ontologies into enterprise application environments grows, new audiences have to deal with ontologies beyond the traditional disciplines that have done so in the past, such as knowledge engineers and ontologists. These audiences range from business users, who need to take ownership of the ontologies, to end users, such as customers or citizens, who are presented with the services based on these ontologies. As the formalisms themselves are often inaccessible to these new audiences, appropriate visualisations are important. Our experience in practice is that business users often overcome their perception of graph oriented visualisations being too technical when gaining experience, but they remain a challenge for incidental reviewers and end users. Therefore, verbalisation of ontologies into natural language is one of the approaches that is crucial to make ontologies accessible to new audiences.

Being able to provide verbalisation in a multilingual manner is important: Governments and enterprise often offer their products and services in international contexts or to customers of different languages. For instance, Dutch Immigrations offers many of its services based on ontologies [1], and it typically needs to interface with people that do not speak Dutch. Also, governments have to deal with numerous international aspects in legislation when drafting their national laws. Specifically in Europe, large parts of national legislation is either heavily influenced by or originates in European legislation. Sharing ontologies capturing such international legislation and being able to refer to them from local ontologies offers important benefits in areas of productivity and traceability across local practices.

In this paper, we present an approach for ontology verbalisation based on the Grammatical Framework (GF) and the Lexicon Model for Ontologies (*lemon*) that is essentially multilingual and addresses the particular challenges when classes are chosen up front based on consensus, while multiple parties contribute often changing instances individually.

We use the Be Informed business process platform[5] as an example throughout this paper. The verbalisation examples described in [2] are built upon and we show how they can be generated across languages transparently. Finally, we illustrate how the challenges in this example generalise across other examples, including the generic case of verbalising Linked Open Data.

2 Related Work

Ontology Lexicalisation The *lemon* model builds on previous research for designing lexica for interfacing with ontologies, in particular that of the LexInfo [3] and LIR [4] models, as well as existing work on lexicon (meta-)models, in particular the Lexical Markup Framework (ISO-24613:2008) [5]. In addition, it builds on efforts to link resources via the Web to the ISOcat meta-data registry [6]. *lemon* seeks to scale the modelling of lexical and linguistic information related to concepts in ontologies from very simple to quite complex lexical entries. *lemon* is closely related the SKOS project[6], which attempts to model simple knowledge organisation systems such as thesauri, classification schemes and taxonomies on the Semantic Web.

Ontology Verbalisation and CNLs The application of CNLs for ontology authoring and instance population is an active research area. *Attempto Controlled English*[7] (ACE) [7], is a popular CNL for ontology authoring and generation. It is a subset of standard English designed for knowledge representation and technical specifications, and is constrained to be unambiguously machine-readable into DRS - Discourse Representation Structure, a form of first-order logic. WYSIWYM (*What you see is what you meant*)[8], involves direct knowledge editing

[5] `http://www.beinformed.com/`
[6] `http://www.w3.org/2004/02/skos/`
[7] `http://www.ifi.unizh.ch/attempto/`

with natural language directed feedback. A domain expert can edit a knowledge base reliably by interacting with natural language menu choices and the subsequently generated feedback, which can then be extended or re-edited using the menu options. With respect to GF, there have been a number of GF grammars using ontologies/databases, such as the grammar representing the structure of SUMO, the largest open-source ontology [8] as a type-theoretical grammar and verbalise it in 3 languages [9]. Moreover, an ontology describing paintings and a database describing over 8,000 artifacts from the Gothenburg City Museum were used for generating descriptions of the paintings in 5 languages [10].

3 Challenges in Verbalising Modular Ontologies

3.1 New audiences for ontologies

An example where ontologies are used in business applications is Be Informed's business process platform. It allows representatives of many disciplines and domain experts to capture the definitions and constraints that the business process must meet and have engines infer executable business processes and decision services from them automatically. This provides business users with a large degree of control and agility, as the constraints are easily changed and the process is updated accordingly. The resulting processes are highly contextual and deal well with exceptions: They allow professionals to influence their own work, and are resilient to the effects of overrides as is described in [2].

When ontologies are used to infer business processes in this way, its stakeholders get to interact with the underlying ontologies on a number of levels. The first is obviously understanding the content of the ontology and the consequences for the inferred business process. The ontology becomes part of the system documentation and is used for validation and reviewing at specification time or for reference when using and maintaining the services and its specification/documentation. Also, the ontology driven services have user interfaces that provide dialog to its end users and the ontology typically is the source of many of the textual elements in that dialog, such as the questions asked, the captions in forms, etc. Finally, the end results of the ontology driven services are also based on the underlying ontology. In applications, these results might be represented by generated documents or letters sent to customers. Specifically when decisions are taken based on models, explaining the end result and the underlying argument have strong ties with the concepts and their textual representations in the ontology.

Be Informed's business processes are inferred from an ontology capturing all relevant activities, artifacts, involved roles etc. and the conditional relations between these. The meta model, as shown in Figure 1, contains the conceptualisation of the business process domain, that instead of using flow semantics to capture who does what and when, is based on pre- and post-conditions. Each activity may have pre-conditions that have to be met before it may be performed,

[8] http://www.ontologyportal.org/

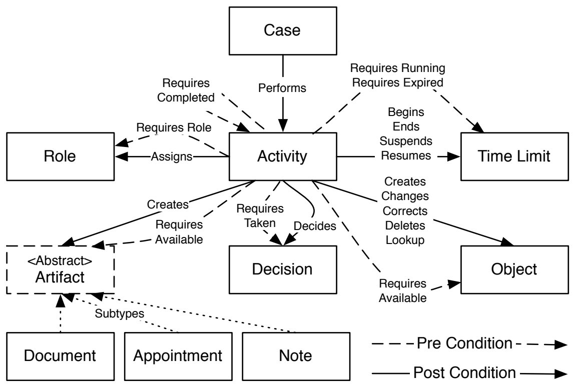

Fig. 1. Summary of the classes used to capture business processes

and post-conditions that capture what conditions have to be met in order for the activity to be complete. Figure 2 contains an of example how this meta model is used to capture the business process of applying for a grant.

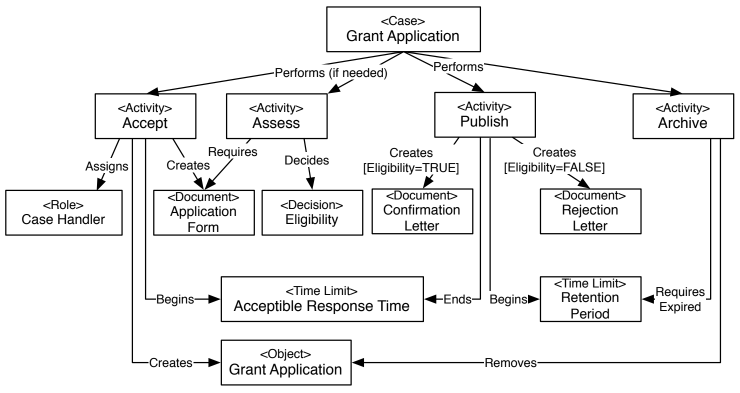

Fig. 2. An example of an grant application process

In [2], this example is discussed in more depth and the model from Figure 2 is verbalised using a pattern sentence approach. The result when verbalising the example used throughout the paper with the pattern sentence approach is the following.

1. The activity Publishing the Result may be performed if
   (a) a document of type Document with Details is available.

2. The activity PUBLISHING THE RESULT is completed if
   (a) a document of type SUBMISSION FORM has been created.

Although the sentences generated have proven useful in practice, some additional requirements exist in areas such as grammatical correctness and fluency. For instance, the first sentence part is grammatically incorrect, as the label inserted is a verb phrase in itself. Also, the pattern sentences approach proves not to scale in terms of number of supported languages. Every grammar translation needs to deal with all lexical aspects (and their peculiarities) of their specific language.

3.2 Differences between classes and instances

Within the field of knowledge representation, both classes and instances are typically treated as equal means of expression when creating an ontology and both can be used by the knowledge engineer as he sees fit to best capture the conceptualisation of his choice.

In the use case presented in this paper however, class and instance information are typically in separate, but linked ontologies and these ontologies differ greatly in characteristics related to ownership and rate of change:

– Classes are chosen based on consensus across multiple parties, while the instances are provided by individual parties without requiring consensus on the data;
– The classes are often determined once and are fixed, whereas the instances of these classes are introduced over time and may change often;
– Classes may use an ontology formalism as its native/original representation, where instances often have existing databases or other information stores as primary sources, and the ontology is used exclusively for sharing the data;
– The classes are typically created by knowledge engineers or other information professionals, while instances are created by a wide range of people, who enter data in the original databases, and might not deal with ontologies at all.

In the Be Informed use case provided throughout this paper, the product contains default meta models consisting of the classes that are used in modelling. Although these might be adjusted or extended in individual implementations, this is typically done early in the design phase. An example of such a meta model is shown in Figure 1. When adopting the product, the majority of effort is spent on creating detailed models based on the classes in the meta model. Meta models and their extensions are typically created by knowledge engineers, while the models themselves are built by both knowledge engineers and domain experts. Moreover, it is considered crucial that business users, who adopt the complete model for validation, can make changes and updates.

The choices made in the lexicalisation/verbalisation of classes and instances respectively, follow a similar pattern as the classes and instances themselves:

– In general the labels and lexical representations for the classes are created alongside the ontology that contains the classes and may require guidelines

to be met. The labels of the instances are often chosen by people less or unskilled in knowledge representation or may even originate from existing databases. This compromises the coherent adherence to guidelines for the lexical properties of instance labels;
– Classes may have complex lexical and grammatical consequences, and also introduce sentence patterns and planning. Instances have only labels with limited complexity at a lexical level.

The separation between reaching consensus on the types of things considered and identifying the individual things generalises well across other scenario's of both the use of ontologies and their verbalisation. For instance, ontologies used in Linked Open Data typically expose these characteristics. An example of this is the verbalisation of structured knowledge about 8000 artifacts from the Gothenburg City Museum into natural language descriptions [10]: The conceptualisation of painters, paintings and materials is codified in an ontology with classes, the facts about the individual paintings are exported out of a database into ontologies with instances.

3.3 Practices in choosing labels

In practice, different styles of choosing labels are found when modellers are, for instance, modelling and naming concepts within a business process model.

In practice, there is a difference between concepts that are commonly referred to by a proper name (term) and concepts that do not have a term associated with them and are referred to by description. For example, when modelling business processes, the ontology typically contains activities. Some activities may have names (terms), but more often a label describes what is done in the activity. Examples are "Publishing the result", "Publish the result" or "The result is published" (when the activity is completed). This phenomenon also occurs in multilingual contexts when a name for the concept exists in the source language of a model but not in the target language, in which case some form of description is necessary.

Another source of label choosing practices can, especially in business use of ontologies, be found in the diverse background of the people involved in modelling. Typically, the domain experts may have experience in other structured conceptualisation approaches and the naming conventions or practices that come with it. For instance, many information professionals have backgrounds in systems development disciplines and are familiar with techniques like UML and Entity Relational modelling, influencing their naming practices.

Providing guidelines and practices for systematically choosing good labels contributes to reducing the number of label variants used in modelling. However, for industry adoptability and robustness in practice, ontology verbalisation techniques should be able to deal systematically with these challenges and practices, both in terms of the number of constraints required for accurate modelling and also the implications of language variation and multilingualism for any given ontology concept.

4 Ontology Verbalisation using GF and *lemon*

We present a multilingual ontology verbalisation approach that addresses the challenges discussed in Section 3. It is based on the Grammatical Framework, currently developed in the Molto Project[9], and *lemon*, the Lexicon Model for Ontologies, currently developed in the Monnet Project[10].

Both projects have a strong focus on ontologies and multilingualism, and complement each other well in our approach: GF provides sophisticated multilingual grammar support while *lemon* provides state of the art ontology lexicalisation techniques.

4.1 Introduction to *lemon* and GF

Lexicon **M**odel for **On**tologies, or *lemon* is a model for representing lexical information about words and terms relative to an ontology on the Web. *lemon* is what we term an *ontology-lexicon* in that it expresses how the elements of the ontology, i.e. classes, properties, and individuals, are realised linguistically. The model follows a principle called *semantics by reference* whereby it is assumed that the (lexical) meaning of the entries in the lexicon are expressed exclusively in the ontology. Hence the lexicon merely points to the appropriate concepts. *lemon* is designed to be a basic model supporting the exchange of ontology-lexica on the Semantic Web. The core of *lemon* contains the basic elements required to define lexical entries and their association to their lexical forms and, moreover, to concepts in the ontology representing their meaning. It consists of:

– **Lexicon:** This object represents the lexicon as a whole. It must be marked with a language, and all objects in the lexicon belong to this language.
– **Lexical Entry:** An entry for a given lexicon is a container for one or several forms. It also contains one or more meanings of a lexeme. All forms of an entry must have the same part of speech. An entry may have multiple meanings.
– **Lexical Form:** This is the inflectional form of an entry. It must have one canonical form, but may have any number of other forms. Stems and other partial morphological units can also be modeled as abstract forms.
– **Representation:** A lexical form may have several representations, ranging from different orthographies, to phonetic representation as well as standard written representation.
– **Lexical Sense:** This links a lexical entry to the **reference** in the ontology i.e. a concept, property or instance.
– **Component:** A lexical entry may also be broken up into a number of components.

The following example gives a simple lexicon with a single lexical entry:

[9] `http://www.molto-project.eu/`
[10] `http://www.monnet-project.eu/`

```
@prefix lemon: <http://www.monnet-project.eu/lemon#>.
@prefix bpo: <http://www.beinformed.com/resource/>.

:lexicon lemon:entry:adult_applicant;
  lemon:language "en".

:adult_applicant
  lemon:canonicalForm [lemon:writtenRep "Applicant is Adult"@en];
  lemon:sense [lemon:reference bpo:adult_applicant].
```

This simple English lexicon has a single entry, with canonical form "Applicant is Adult", and a sense that refers to the entry in Be Informed Business Process ontology.

The **G**rammatical **F**ramework (GF) [11] is a formalism for describing multilingual grammars. A GF grammar consists of an abstract syntax, acting as a semantic interlingua and a number of concrete grammars that verbalise the abstract syntax in multiple languages. In this way the semantic level is the central part of the grammar, connecting any language pair of concrete syntaxes.

Most GF grammars are used to describe fragments of natural language. The largest and most general such grammar is the resource grammar library, which contains *resource grammars* for 24 languages, and implements the most common syntactic constructions (such as predication, complementation, etc.) for these languages.

The resource grammar library supports the development of so-called *application grammars* for more restricted domains by providing standard language-specific technicalities so that they do not need to be described again in the new (application) grammar. This makes it easier to develop application grammars by assembling the primitive constructs from the resource grammar and obtain syntactically-correct text in languages from the library.

In the work reported on in this paper GF is used to develop an application grammar for the Be Informed use case, discussed in section 3.

4.2 Modular GF grammars based on decoupling of classes and instances

The ontology verbalisation approach exploits the complementary strengths of GF and *lemon*. GF is used to capture ontological information as well as the required sentence structure while *lemon* is the source of concrete label information. The approach is explained by using as example the Be Informed business process ontology and the pre- and post-condition sentence patterns of [2]. The aim of the business processes ontology in Figures 1 and 2 is to specify business processes in terms of pre- and post-conditions, which in turn requires the verbalisation of such conditions.

A challenge resulting from the strict separation of (ownership of) TBox and ABox is that the lexical information required in verbalisation of the complete ontology also needs modularisation along these boundaries and that restrictions with respect to availability/feasibility of knowledge engineering to the different ontology parts also apply to the lexical information associated with those parts.

The grammar modularisation follows the structure of the meta model in Figure 1 (a similar approach was followed in [10]). The verbalisation proceeds according to the sentence patterns described in [2]. In particular, each activity may have pre-conditions and post-conditions verbalised as *conditional statements* ("A if B"), where A and B are simple *propositional statements* with modalities, as appropriate. All *concept labels* are to be verbalised as propositional statements in accordance with specifications, as explicated in [2].

The modular layered approach to verbalisation is illustrated in Figure 3 by means of the pre-condition triple

```
(Activity,Requires_Available,Artifact subtyped as Document).
```

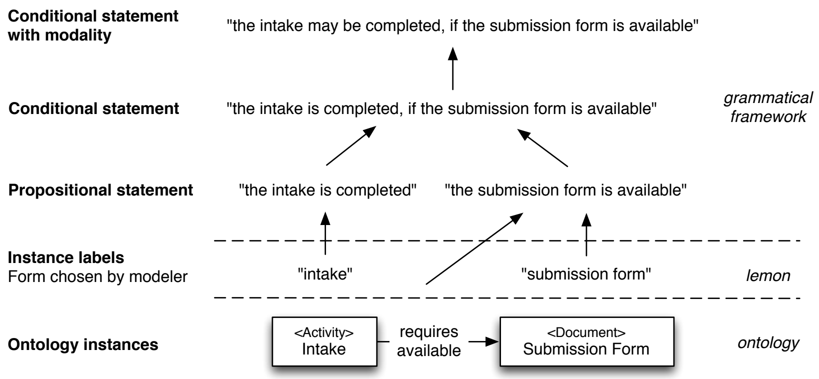
Conditional statement with modality
"the intake may be completed, if the submission form is available"
Conditional statement
"the intake is completed, if the submission form is available"
grammatical framework
Propositional statement
"the intake is completed"
"the submission form is available"
Instance labels
Form chosen by modeler
"intake"
"submission form"
lemon
Ontology instances
<Activity>
Intake
requires available
<Document>
Submission Form
ontology

Fig. 3. Layered verbalisation procedure

The TBox abstract syntax caters for the pattern sentence structure (the conditional and propositional statements), the modalities, the concept classes and the relations between them. The polarities and modalities are introduced as separate functions, allowing for the addition of more modalities by merely creating two additional functions per modality (positive and negative form), using the primitive operation already defined. The (meta model) classes are modelled as GF categories and the relations become GF functions with a return type used for verbalisation, as shown in the code segment taken from the TBox abstract syntax module:

```
cat
 Activity;  -- meta model type
 Document;  -- meta model type
 Artifact;  -- meta model type
```

```
  Fragment;  -- type for proposition
  BIText;    -- type for language generation

fun
  requires_available : Activity -> Artifact -> Fragment;
  subDocArtifact : Document -> Artifact;
  FCan : Fragment -> BIText;
```

The conceptual modularisation of the grammar is completed by the ABox abstract syntax, which uses the TBox abstract syntax, and contains the label/instance definitions (ABox information) as GF function declarations with category types, for example,

```
 AIntake : Activity;
 SubmissionForm : Document;
```

The verbalisation of the information captured by the ontology, as well the pattern sentences in which they may occur, is addressed in the concrete syntaxes for English and Dutch. The linearisation categories for the (abstract syntax) categories provide detailed linguistic structure in terms of parts of speech etc, for rendering correct verbalisations. Complex (GF record) types are used to facilitate label variants. The predicates that are built with transitive verbs (creates, deletes, corrects, changes) are rendered in the passive voice, as required by the Be Informed application. The TBox concrete syntax also provides primitives for creating the main categories, for example `mkActivity` (see next section) for `Activity` from basic parts of speech from the resource library, and hides the implementation details from the users, thereby ensuring that the optimisations are as seamless as possible.

The TBox concrete grammar (code segment below) provides the linearisation of both the propositional and the conditional statements. In particular, the overloaded function `mkFragm` implements both the simple propositional statement (as complete sentence pattern) (`hasExt = False`) and the conditional statement (`hasExt = true`) by means of pattern matching on the number and types of the arguments. The overloaded function `mkBIText` completes the verbalisation by finally adding modality and polarity, as necessary. We show `mkFragm` and a part of `mkBIText` by way of illustration.

`Utt` and `S` are GF resource grammar library categories for sentences, questions, etc. and declarative sentences, respectively, while `Fragm` is a user defined complex (GF record) type. `VV` is the the resource grammar library category for verb-phrase-complement verbs.

```
lincat
  Activity  = {noun : NP; subj : NP; vp : VP; hasVerb : Bool};
  Document, Artifact = NP;

lin
```

```
  requires_available ac ar = mkFragm ac.subj ac.vp (mkS (mkCl ar
  (mkVP available_A)));
  subDocArtifact d = d;
  FCan frag = mkBIText frag positivePol may_VV;

oper
  Fragm = {subj : NP; pred : VP; ext : {s : S; hasExt : Bool}};

  ifExt : {s : S; hasExt : Bool} -> S -> S = \ext,s -> case
  ext.hasExt of {
    True  => Sentence.SSubjS s if_Subj ext.s;
    False => s
    };

  mkFragm = overload {
  mkFragm : NP -> VP -> Fragm =
    \np, vp ->
      {subj = np;
      pred = vp;
      ext = {s = dontCareS; hasExt = False}};

  mkFragm : NP -> VP -> S -> Fragm =
    \np, vp, sub ->
      {subj = np;
      pred = vp;
      ext = {s = sub; hasExt = True}};
  };

  mkBIText = overload {
  mkBIText : Fragm -> Pol -> Utt =
    \frag, pol ->
      mkUtt (ifExt frag.ext (mkS pol (mkCl frag.subj frag.pred)));
.......

  mkBIText : Fragm -> Pol -> VV -> Utt =
    \frag, pol, vv ->
      mkUtt (ifExt frag.ext (mkS pol (mkCl frag.subj
      mkVP  vv frag.pred))));
  };
```

In the ABox concrete syntax the labels are defined according to the required types using either provided primitives or basic parts of speech, for example,

```
 AIntake = mkActivity (mkNP the_Quant intake_N);
 DSubmissionForm = mkNP the_Quant (mkCN submission_N (mkNP form_N));
```

where the functions `mkN`, `mkCN`, `mkNP`, `mkVP`, `mkCl`, `mkS`, `mkUtt`, etc., and many more, are available in the GF resource grammar library for supporting and facilitating application grammar development.

The function `mkActivity`, used in linearising the label `AIntake`, converts the label "intake" to a propositional statement, where the verb to be used with activities expressed as nouns or noun phrases is always "to complete', i.e. "the intake is completed" as prescribed by the meta model in Figure 1. We return to the `mkActivity` function in Section 4.3 where the handling of label variants is discussed.

The final (customary) component in the suite of modules that constitute a GF application grammar is a dictionary of application specific lexical items that do not occur in the resource grammar lexicon and additional linguistic constructs that are language specific and/or are not included in the resource grammar, for example, verb nominalisation. Examples (showing the abstract as well as the concrete syntax) are as follows:

```
 oper
  intake_N:N;
  submission_N:N;
  form_N:N;
```

and

```
 oper
  intake_N = mkN "intake";
  submission_N = mkN "submission";
  form_N = mkN "form";
```

An example of a pre-condition triple and its verbalisation in English and Dutch by means of the GF grammar is follows:
(`Activity`, `Requires_Available`, `Artifact`, subtyped as `Document`), instantiated with the labels "intake" and "submission form", is rendered as the pre-condition, containing the propositional statements "the intake may be completed" and "the submission form is available", with modality added:

```
The intake may be completed , if the submission form
is available.
De inname kan worden afgerond , als het aanvraagformulier
beschikbaar is.
```

Other typical examples of linearisations are as follows:

Pre-conditions:

```
44b may be completed , if the document with details
is available.
44b kan worden afgerond , als het document met details
beschikbaar is.
```

Post-conditions:

```
If 44b has been completed , the submission form has been created.
Als 44b afgerond is , is het aanvraagformulier aangemaakt.
```

Propositional statements (intermediate layer):

```
44b has been completed.
44b is afgerond.
```

In summary, the modularisation of the grammar was illustrated by showing how concepts and relations between them are introduced as categories and functions in the TBox abstract syntax and reused in the in the ABox abstract syntax to instantiate abstract instances. The concrete syntaxes were shown to provide linearisations, using different variants, for the instance labels, the propositional, and the conditional statements that consitute the pre- and post-conditions in the ontology.

4.3 Dealing with different variations of labels

In the previous section the focus was on modularisation as a mechanism to support efficient and effective ontology development. By separating the TBox and ABox information in different grammar modules the modeller is allowed to concentrate on application dependent information while generic business process modelling support is provided by the grammar. In this section we discuss the extent to which the BI grammar may allow for label variants and what kinds of variation is accommodated at present.

Basically, the grammar allows two broad kinds of labels. Firstly labels in the form of nouns or compound nouns (names or terms) are permitted, for example, "Intake" and "Equality principle". Secondly, labels may take the form of a proposition such as "The Result is published" or other verb oriented style labels such as "Publishing the result" or "Publish the result".

While allowing the modeller some freedom of choice in label selection, the verbalisation of label variants that refer to the same concept or relation in the ontology should be unique.

The following code fragment from the TBox concrete syntax shows how some of these design choices may be implemented:

```
mkActivity = overload {
mkActivity: N -> V2 -> NP -> Activity = \n,v,o -> lin Activity {
   noun = mkNP the_Quant (mkCN n (mkAdv of_Prep o));
   subj = o;
   vp = passiveVP v;
   hasVerb = True
};
mkActivity: V2 -> NP -> Activity = \v,o -> lin Activity {
   noun = nominalize (mkVPSlash v) o;
```

```
    subj = o;
    vp = passiveVP v;
    hasVerb = True
 };
 mkActivity: NP -> Activity = \o -> lin Activity {
    noun = o;
    subj = o;
    hasVerb = False
 }
};
```

In this way, the ABox concrete syntax may be used to create an object of type `Activity` by, for example, either providing an `NP` for simple cases like "Intake" or a `V2` or an object `NP` for labels such as "Publishing the result" and "Publish the result".

There are a number of other fields that are used in the English TBox concrete syntax that ensure an optimized natural language generation such as "The Results are published" or "Intake is completed if the results are published", but these details are just handled in the grammar, and the users of the ABox syntax do not need to be aware of them.

It should be noted that these choices of implementation are, to some extent, language specific since in labels of the latter kind the English gerund is used while in Dutch the infinitive form of the verb plus "van" is customary.

In the following example the label variant "publish the result" is rendered as in the sentences below:

Post-condition triple:

```
(Activity, Creates_Artifact, Document)
```

Linearisation:

```
If the result has been published , the submission form
is available.
Als het resultaat gepubliceerd is , is het aanvraagformulier
beschikbaar.
```

4.4 Multilingual aspects

In the GF grammar the multilingual correspondence of the bilingual system is via the ontological labels. For instance, "Publishing the result" in English corresponds to "Publiceren van het resultaat" in Dutch because they both map to the instance `APublishingOfResult`. However, word-wise there is no one-to-one correspondence, since this would not scale up when adding more languages and extending the ontology with more instances. From this perspective it resembles the *lemon* notion of multilingualism and supports the automated mapping from *lemon* to GF, as discussed in a subsequent section.

Moreover, since the TBox syntax mainly uses the resource library and aligneable primitives from there, it can be abstracted in a functor, so that the new languages can inherit it, at least partially. In this way, adding a new language would be a simpler process, since it would mainly require the writing of a new lexicon and translating the instances from the ABox concrete syntax. An investigation into this aspect forms part of future work.

4.5 Generating the instance grammars from *lemon*

We reiterate that the exploratory work presented in this paper focuses on three aspects of modular ontology verbalisation, as also illustrated in Figure 3. In section 4.2 we addressed the modularisation of the GF grammars, used for the verbalisation, in accordance with the modular ontologies. It was noted that the labour intensive component of the product is the creation of the ABox grammars (that contain instances and their labels) and that the flexibility in specifying labels is of strategic importance. Section 4.3, therefore, focused on label variants currently allowed by the GF grammar. In this section we now briefly turn our attention to the third aspect viz. an automated way of populating the ABox grammars by making use of *lemon*. The approach is based on the observation that the *lemon* entries contain all the essential information required to construct GF lexicon entries and linearisation rules for the instance labels in the ABox grammars.

We illustrate this by means of an example in which essential parts of the linguistic information, necessary to create an instance label, are shown. We consider the entry for the instance label "Sketch of the situation", which is represented as a *lemon* `decomposition`:

```
 #Sketch of the situation
lemon:decomposition ( :sketch_component :prep_component
                :det_component :sit_component );

lemon:phrase_Root
[ lemon:constituent :NP;
 lemon:edge [ lemon:constituent :NN; lemon:leaf:sketch_component];
 lemon:edge [ lemon:constituent :PP;
  lemon:edge [ lemon:constituent :P; lemon:leaf:prep_component];
  lemon:edge [ lemon:constituent NP;
   lemon:edge [ lemon:constituent :DET; lemon:leaf:det_component];
   lemon:edge [ lemon:constituent :NN; lemon:leaf:sit_component
         ]]]];

:sketch_component lemon:element sketch.
:sketch a lemon:Word .
:sketch isocat:partOfSpeech isocat:noun .

:prep_component lemon:element of.
```

```
:of a lemon:Word .
:of isocat:partOfSpeech isocat:preposition .

:det_component lemon:element the.
:the a lemon:Word .
:the isocat:partOfSpeech isocat:determiner.

:sit_component lemon:element situation.
:situation a lemon:Word .
:situation isocat:partOfSpeech isocat:noun .
```

It consists of a `phrase_Root` entry which describes the syntactic decomposition of the entry. The *lemon* entry is decomposed into a noun phrase, in turn decomposed into a noun, a preposition, a determiner and another noun. Each component in the *lemon* decomposition is comprised of `elements`, for example the preposition `of` a the noun `situation`.

In GF the same information is encoded as follows, making use of functions from the resource grammar library:
ABox abstract syntax:
`DSketchOfSituation : Document` where `Document` is a class in the ontology and a category in GF.
ABox concrete syntax:
`DSketchOfSituation = mkNP a_Quant (mkCN sketch_N (PrepNP of_Prep (mkNP the_Det situation_N)))` where `mkNP`, `mkCN` and `PrepNP` are functions from the resource grammar library.
Dictionary abstract syntax: `sketch_N : N` and `situation_N : N`
Dictionary concrete syntax: `sketch_N = mkN ''sketch''` and `situation_N = mkN ''situation''`.

The constituent structure (syntax tree) of the label "Sketch of the situation" is given in the first part of the *lemon* entry and by the GF ABox concrete syntax entry, while the lexical information is available in the second part of the lemon entry and the GF dictionary entry. An appropriate generalisation of this correspondence (mapping) will cover all possible *lemon* entry syntax trees and their GF equivalents. This will form the basis for an automated procedure to generate ABox concrete syntax entries for instance labels from *lemon* and is part of our future work. Furthermore, this approach is well suited to multilingual entries in *lemon* and their representation and verbalisation in GF, the investigation of which also forms part of future work.

5 Discussion and Future Work

In this paper we demonstrated how GF and *lemon* can be combined to provide multilingual verbalisation in a specific class of problems, where ontologies are modularised into ontologies containing consensus based classes and (multiple) instance ontologies. In these scenarios, the instances are typically not created by knowledge engineers and are often based on information stores other than ontologies.

We showed an approach to grammar development that leads to grammars being modular along the same boundaries as the ontologies, with the grammar modules also having the same dependency structure as the ontology parts.

Additionally, we introduced mechanisms in the TBox grammar that deal automatically with the different styles of label choosing that are encountered in practice. This is particularly relevant in cases where different disciplines are involved in creating the instances and in multilingual cases where concepts may have a name in one language and can only be referenced by describing them in another language.

By generating the instance grammars from the ontology labels encoded in *lemon*, the need for GF engineering at the instance level is reduced. This is crucial for the adoption of this mechanism, as the instance data either already exists in databases or is created by people who may not be expected to engineer GF representations of the labels they choose.

In terms of multilingualism this approach supports and suggests two ways of verbalisation. In cases where translations of the labels are available inside *lemon* lexica, multiple ABox syntaxes can be generated for the different languages. This approach has the benefit that labels may differ quite significantly across languages, but obviously requires translation of the ontology. Alternatively, translation could be performed at the GF grammar level if the label styles align across languages. In cases where aligned dictionary grammars are available, this approach could prove particularly efficient.

Future work also includes the extension of this approach to include different levels of paraphrasing and advanced sentence planning in order to achieve improved fluency in and across sentences. Finally, we envisage investigating label choosing practices as encountered amongst professionals in the different fields that may benefit from a framework as described in this paper. The importance of robustness under lexical variance as found in real world applications suggests an in depth study of label style variation and its impact on ontology verbalisation.

Acknowledgements The authors wish to gratefully acknowledge the support for this work by the European Commission (EC). This work was partially funded by the EC within the EU FP7 Multilingual Ontologies for Networked Knowledge (MONNET) Project under Grant Agreement No. 248458 and the MOLTO Project under Grant Agreement No. 247914.

References

1. Heller, R., Teeseling, F.: Knowledge applications for life events: How the dutch government informs the public about rights and duties in The Netherlands. In: Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications. ESWC 2009 Heraklion, Berlin, Heidelberg, Springer-Verlag (2009) 846–850

2. Van Grondelle, J.C., Gülpers, M.: Specifying flexible business processes using pre and post conditions. In: PoEM. Volume 92 of Lecture Notes in Business Information Processing., Springer (2011) 38–51
3. Buitelaar, P., Cimiano, P., Haase, P., Sintek, M.: Towards linguistically grounded ontologies. In: The Semantic Web: Research and Applications. (2009) 111–125
4. Montiel-Ponsoda, E., de Cea, G., Gómez-Pérez, A., Peters, W.: Modelling multilinguality in ontologies. In: Proceedings of the 21st International Conference on Computational Linguistics (COLING). (2008)
5. Francopoulo, G., George, M., Calzolari, N., Monachini, M., Bel, N., Pet, M., Soria, C.: Lexical markup framework (LMF). In: Proceedings of the 2006 International Conference on Language Resource and Evaluation (LREC). (2006)
6. Kemps-Snijders, M., Windhouwer, M., Wittenburg, P., Wright, S.: ISOcat: Corralling data categories in the wild. In: Proceedings of the 2008 International Conference on Language Resource and Evaluation (LREC). (2008)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web, Fourth International Summer School 2008. Number 5224 in Lecture Notes in Computer Science, Springer (2008) 104–124
8. Power, R., Scott, D., Evans, R.: What you see is what you meant: direct knowledge editings with natural language feedback. In Prade, H., ed.: 13th European Conference on Artificial Intelligence (ECAI'98). John Wiley and Sons, Chichester, England (1998) 677–681
9. Angelov, K.A., Enache, R.: Typeful ontologies with direct multilingual verbalization. In: Controlled Natural Languages Workshop (CNL 2010), Marettimo, Italy (2011)
10. Dannélls, D., Enache, R., Damova, M., Chechev, M.: Multilingual online generation from semantic web ontologies. In: WWW2012. EU projects track, Lyon, France (04/2012 2012)
11. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).