D12.2 User studies for BI's explanation engine

Contract no.	FP7-ICT-247914
Project full title:	MOLTO - Multilingual Online Translation
Deliverable:	D12.2 User studies for BI's explanation engine
Security (distribution level):	Public
Contractual date of delivery:	31 May 2013
Actual date of delivery:	31 May 2013
Type:	Report
Status & Version	Draft, 0.9
Author(s)	Joris van Aart, Jeroen Daanen, Jouri Fledderman, Jeroen van Grondelle, Menno Gulpers, Emiel van Haandel, Herko ter Horst, Frank Smit, Xander Uiterlinden
Task Responsible	Be Informed

Abstract

This document outlines the evaluation results from the verbalization techniques adopted in the verbalization component for the Be Informed Business Platform based on MOLTO Technologies in WP12 of the MOLTO project. First this document will focus on the evaluation of the adoption of GF technologies within our development department. Secondly results of the actual verbalizations will be presented and discussed.

Table of Contents

1 Introduction 2
1.1 Background 2
1.2 About this Document 3
1.3 Contributors 3
2 Adoption of Grammatical Framework (GF) in Be Informed 4
2.1 Phase 1: Education 4
2.2 Phase 2: Application 5
2.2.1 3D model 5
2.2.2 Grammars 6
2.2.3 Verbalizers 9
2.3 Phase 3: Proposed development model 11
3 Evaluation of Verbalization Techniques based on GF 13
3.1 Background and Evaluation Methodology 13
3.2 Evaluation Methods 13
3.3 Experimental Setup 13
3.4 Results and Discussion 14

1 Introduction

1.1 Background

As the adoption of ontologies into enterprise application environments grows, new audiences have to deal with ontologies, other than 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 visualizations are important. Our experience in practice is that business users often overcome their perception of graph-oriented visualizations being too technical when gaining experience. However, graph visualizations remain a challenge for incidental reviewers and end users. Therefore, verbalization of ontologies into natural language is one of the approaches that is crucial to make ontologies accessible to new audiences.

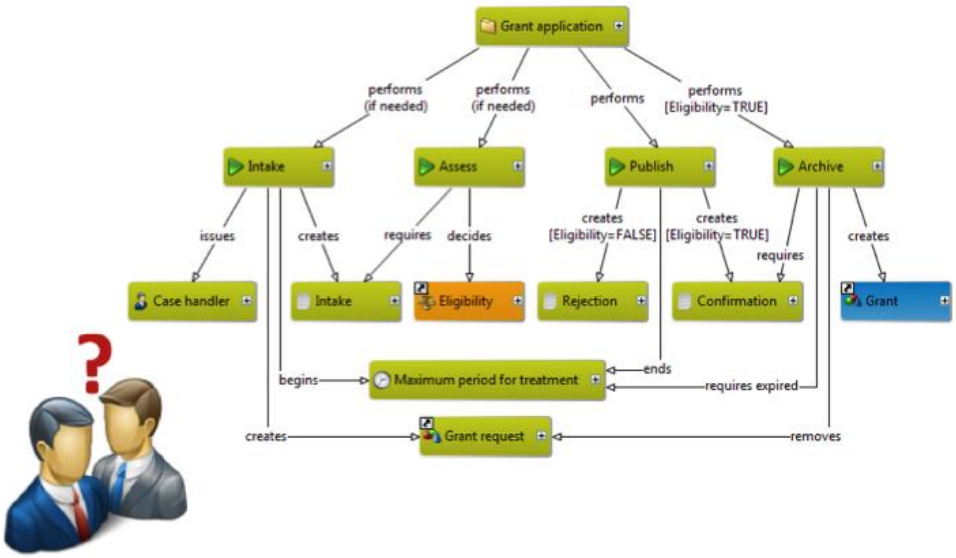

Figure 1. Poor Business User Adoption of Graphical Visualisations

Additionally, being able to provide verbalization 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 [ESWC2009], 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 are either heavily influenced by or originates in European legislation. Being able to share 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 2010 Be Informed has developed a verbalization component based on pattern sentences, that is released as part of our product. It is discussed in detail in [EKAW2010] and [CNL2010].

The areas that need improving outlined in specifically [CNL2010] triggered our participation in the MOLTO Project.

1.2 About this Document

This document outlines the requirements that we will need to address when developing a verbalization component for Be Informed based on MOLTO Technologies in WP12 of the MOLTO project.

We have chosen a broad, slightly informal style of requirement capturing. We believe it improves readability and will make the document relevant for broader audiences. We have tried to capture requirements from a large number of perspectives. Some requirements apply to the verbalization component to be developed in WP12, but many also apply to the functionality that can be based on this component. Although out of scope for WP12, we believe it is the best way to visualize intended use and capture the inherently implicit requirements that this might pose on a technology we do not completely master at this time.

No formal distinction between must have and optional requirements is made. We believe the document will guide us in leveraging GF to the maximal extend in the development of a verbalization component in WP12.

We will use it for WP12 planning and resourcing, both within Be Informed and in discussions with Chalmers University concerning its role in WP12.

We will also use it when designing the verbalization component based on GF and the grammars for our four default modeling domains.

This document does not contain a detailed design of the grammars or the verbalization component, but rather the requirements the grammars should address.

1.3 Contributors

Editor of this document is Jeroen van Grondelle.

Contributing authors are Joris van Aart, Jeroen Daanen, Jouri Fledderman, Jeroen van Grondelle, Menno Gulpers, Emiel van Haandel, Herko ter Horst, Frank Smit and Xander Uiterlinden.

Please direct questions, contributions and ideas to Jeroen van Grondelle at j.vangrondelle@beinformed.com.

2 Adoption of Grammatical Framework (GF) in Be Informed

2.1 Phase 1: Education

With proper education, like a tutor who already knows a lot about the Grammatical Framework, it takes about a day or two to enable someone to write a grammar from scratch in GF. The online manual usually provides the necessary information and if not, the book about GF probably will. Together with the website, the book and the resource grammar libraries it is then possible to create an abstract grammar and multiple concrete grammars for different languages. Even if you lack the grammatical knowledge about a particular language GF is able to generate grammatically correct sentences. This is where you see the Grammatical Framework in its full power, it is fast, reusable and works as intended.

However, upping the level and thus writing some more complex grammars, lets you experience the boundaries of the GF as well, for example the fact that it has a very small community. Information on very specific topics, such as dependent types, is very limited on the website as well as in the book. Also the error report sometimes fails to point to the right direction in these situations. The nice thing is that in these situations the small, but very dedicated, community is very willing to help out. The community usually provides an answer within a day, but this means at the same time that a day is lost waiting for the answer.

For writing grammars we made use of a text editor and for testing them we used the GF shell. The reason that we did not use any tooling provided by the community was that we either did not know of their existence or we did not see an advantage of the tool over the text editor and the shell. Our goal was to create grammars, build PGF's and linearize AST's directly from our JAVA code. Linearizing AST's can be done by using JPGF with an predefined PGF. However because we create grammars on the fly a PGF should be built from these grammars. Thus the JPGF library was not suitable for our purposes. Therefore we decided to use the GF-JAVA library, created by Kaarel Kaljuurand, which allowed us to communicate with a GF server (either locally or in the cloud) directly from JAVA code.
However, for bandwidth and memory purposes we were asked not to use the cloud server, so for that reason we use have to start a local GF server.

This means that the learning curve for GF is low in the very beginning, since it is fairly easy to start writing grammars and there is plenty of documentation to help you getting started. However, this curve gets exponentially higher with more complex grammars. The reason for this is that the point where the grammars are getting more difficult to write is

the same point where the error handling is providing less clear pointers and the documentation only covers the basic ideas and not everything in specific.

2.2 Phase 2: Application

The idea behind the application was to automatically create verbalizations from ontologies. While domain experts have the knowledge about a certain law or procedure, they usually lack the knowledge and skill in reading a Be Informed model. By creating verbalizations from our models, we provide the domain experts with an easy way to validate if a law or procedure is modeled correctly. Furthermore, since these verbalizations are created by filling in the different parts of a triple into certain slots of a verbalization, a triple can thus be seen as a reusable proposition. This means that a set of triples used with validation linearization categories and functions can be used to generate validation sentences of the model, while the same set of triples in combination with explanation linearization categories and functions can also be used generate sentences that explain the model.

In order to be as reusable as possible, the purpose was not only to verbalize Be Informed ontologies, but to create a general framework to verbalize ontologies. Choices regarding this reusability are for example: creating the 3d model together with the university of Bielefeld model (explained more thoroughly later this paragraph), using the OWL ontology format besides our own Be Informed ontologies and using open source tools such as LeMOn, which was created by the Monnet project (John McRae) and Lemon2GF (Christina Unger).

2.2.1 3D model

Together with the university of Bielefeld the 3D framework (figure 1) was set up (Van Grondelle & Unger, 2013). This 3D framework states that verbalizing a certain triple is influenced by three dimensions:

- **Domain** (business rules, travel, weather)
- **Task** (query, dialog, explanation, validation)
- **Language** (English, Dutch, French)

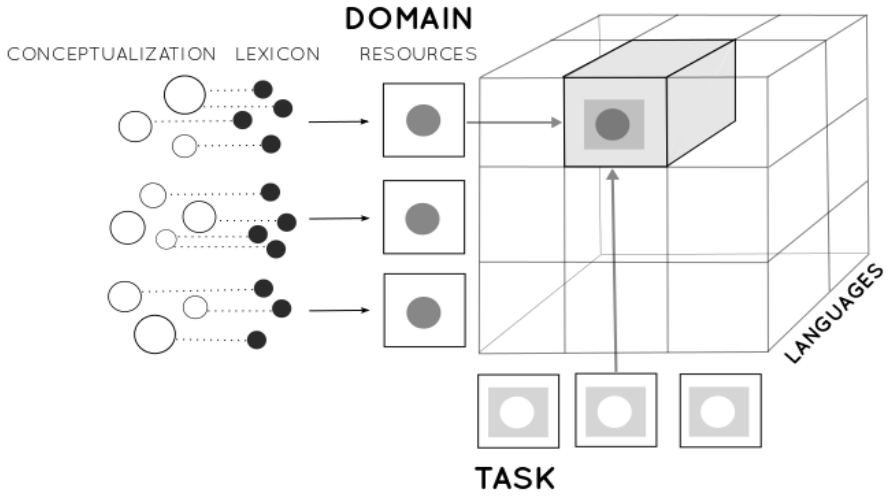

Figure 2: Three dimensional model for conceptually-scoped language technology

This orthogonal modularization supports specification of the conceptualization and lexical information per dimension, i.e. specifying domains independent from tasks and vice versa. The dimensions can then be freely combined by choosing the particular domains, tasks and languages supported for a specific application. This allows not only for the reuse of already existing conceptualizations, such as adding new tasks to an existing domain or reusing task conceptualizations across different domains, but steadily increases the return on investment, since the more of these building blocks already exist, the easier and faster it is to plug them together to build new applications.

2.2.2 Grammars

While the 3D model states that the dimensions involved in verbalizing a triple are domain, task and language, the grammars used to verbalize a triple are not one to one reflections of the dimensions, as can be seen in figure 2. While the language dimension is not represented as a specific grammar, this figure shows core grammars which were not specified as a certain dimension.

The reason that the language dimension is not represented as a grammar in the framework is because all grammars consist of a functor together with set a concrete grammars for different languages. In this way the Resource Grammar Library played a very important role in porting the grammars to other languages without much effort.

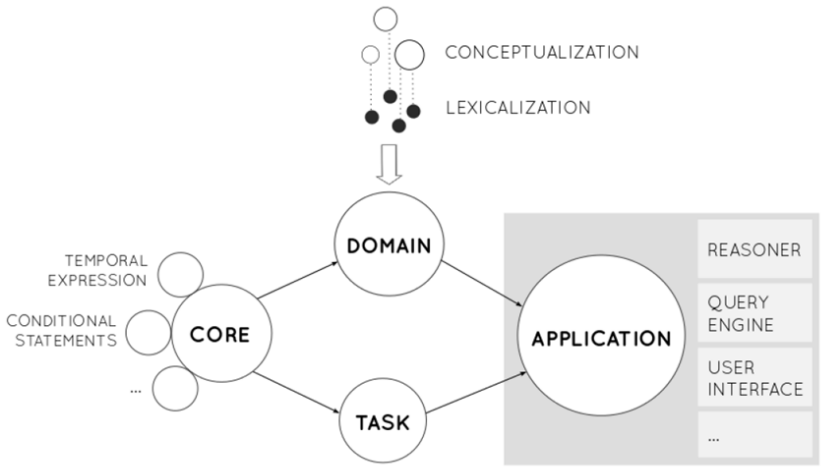

Figure 3: Grammar modularity. Arrows indicate grammar inheritance.

Core Grammars

The core grammar comprises domain- and task-independent expressions, especially closed class expressions such as determiners, pronouns, auxiliary verbs, coordination expressions and negation. It can be extended by libraries that further specify expressions that a domain or a task might rely on, but that are not required in all application, such as temporal expressions or conditional and causal statements. The core grammar is generated manually and can be reused. for every domain and task. It provides an independent basis on which both domain and task grammars build, acting as a decoupler between them. The core grammar is divided into 4 parts: (1) resource grammar, (2) basic grammar, (3) main core grammar, and (4) component libraries. The resource grammar is made available by GF trough the resource grammar API. This grammar is inherited by all other grammars in the library. The second grammar, the basic grammar, is an extension of the resource grammar that contains manually defined operations and parameter types. Figure 3 shows a small part of this so-called basic grammar. Here is shown how a ClassRecord is defined, which could be used as linearization type in any concrete grammar, and how the mkClass function can compute from different argument types a ClassRecord. Furthermore, the basic grammar also contains operation for the NONE type, which is used when that specific value in the record is not specified by the argument types.

```
mkClass = overload {
  mkClass : CN -> ClassRecord = \cn -> {cn=cn; vp=NONE; ap=NONE; adv=NONE};
  mkClass : VP -> ClassRecord = \vp -> {cn=NONE; vp=vp; ap=NONE; adv=NONE};
  mkClass : AP -> ClassRecord = \ap -> {cn=NONE; vp=NONE; ap=ap; adv=NONE};
  mkClass : Adv -> ClassRecord = \ad -> {cn=NONE; vp=NONE; ap=NONE; adv=ad};
  mkClass : CN -> AP -> ClassRecord = \cn,ap -> {cn=cn; vp=NONE; ap=ap;
    adv=NONE};
  mkClass : VP -> VPSlash -> ClassRecord = \vp,vps -> {cn=NONE; vp=vp;
    ap=NONE; adv=NONE};
};
```

Figure 4: Overload function for mkClass

The third grammar, core grammar, defines the core categories and functions The core grammars are complemented with component libraries that contain the categories and functions specific for a specific ontology type. The OWL library for example is an extension of the core grammar that contains OWL specific functions. Such as, a function that expresses an *is-a* relation, functions to express cardinality restrictions, and a function for the owl_Thing.

Domain grammars
The domain grammar extends the core with expressions that are automatically generated from a given ontology lexicon. The domain grammar contains all information provided by the domain knowledge (i.e. domain ontology). Thus it specifies what the classes, individuals, and relations are given the ontology. The abstract domain grammar contains the declarations, for this is uses the categories defined in the core grammar. Therefore, the abstract domain grammar only contains functions and no categories. Table 2 shows how different OWL URIs are transformed to functions in a GF abstract grammar. The table shows that classes are mapped to functions of type Class, individuals are mapped to functions of type Individual, and object properties are mapped to functions from two Individuals to a Statement.

	OWL	GF abstract syntax
Class declaration	http://www.beinformed.nl /owl/ontology#Document	Document : Class
Individual declaration	http://www.beinformed.nl /owl/ontology#Intake	Intake : Individual Activity;
Object property declaration	http://www.beinformed.nl /owl/ontology#Creates	Creates : (c1, c2) Individual c1 → Individual c2 → Statement

Table 1: Mapping from OWL declaration to GF abstract syntax function using some examples. The URI of the declaration is used as function name in GF.

The concrete grammar can be built by using the operations defined in the basic grammar. A written representation of the OWL declaration is needed as a lexicalization of the declaration. Every conversion has its own operations. These are: mkClass for class declarations, mkIndividual for individual declarations, and mkStatement for objec property declarations. Table 3 shows an example of how the class declaration http://www.beinformed.nl/owl/ontology#Document could be written in a concrete syntax form using the word ‘document’ as the lexical form of the declaration. The word `activity' is a Noun. As figure 5 shows, mkClass takes as input a Common Noun (CN). Using the resource grammar, a CN can be created from a N. This is shown by the concrete syntax in table 2.

	OWL	GF abstract syntax
Class declaration	http://www.beinformed.nl/owl/ontology#Document	Document = mkClass (mkCN (mkN “document”));

Table 2: Linearization of Class ‘Document’ in a concrete Be Informed domain grammar

Task Grammars
task grammars extend the core with task-relevant expressions, such as question words and constructions in the case of a querying task. As of now it is created manually, but carrying over the grammar generation pipeline from the domain to the task dimension and thereby also allowing for the automatic generation of task grammars constitutes future work. The tasks on the task dimension are for example: Validation, Explanation, Querying, Online Dialog etc. These separate task grammars all contain of multiple categories and functions so that for each of the tasks multiple linearizations of the same statement can be made. Table 1 shows the example linearizations for a set of tasks for the triple: Intake (Activity) creates ApplicationForm (Document).

Task	Function	Linearization
Query	Yes/No	Does the intake create the application form
Query	QueryAdv	What does the intake create?
Validation	Unless	No application form is created unless the intake is performed
Validation	Generalize	The intake creates a document
Online Dialog	$2^{nd}/3^{rd}$ person	I would like to inform you that the intake form is created
Explanation	Vanilla	The application form is created if the intake is performed

Table 3: Example linearizations for the triple: Intake (Activity) creates ApplicationForm (Document).

2.2.3 Verbalizers
While the core and the task grammars are handmade, the domain grammars are generated automatically. Be Informed has three methods, called verbalizers from now on, of creating a grammar out of an ontology. These methods are called: *Naïve*, *Naïve with heuristic* and *LeMOnAided*. This paragraph discusses each of the verbalizers thoroughly.

Naïve
This process, like the name already suggests, is the most naive process. It is able to turn a Be Informed or an OWL ontology directly into a

verbalization. In order to do this, it takes the labels from the object properties, the classes and the named individuals or the Be Informed equivalents of these entities. In the 3d framework we described Classes, Individuals and Relations which directly map to these OWL entities. While an overload function is specified in the Framework to deal with different labels, this verbalizer does not care for this label variance. It just assumes that all Classes map to common nouns in GF, all named individuals map to proper names and all object properties map to functions that create a statement out of two Individuals.

However, since GF comes with rules on how conjugate verbs and the labels of the Be Informed models without exception contain a third person singular verb in present tense, sometimes complemented with the use of a noun, an adjective or another verb, we encountered a problem. In order for the naïve verbalizer to overcome this problem we created a template containing our Tbox relations. These templates are written by hand once, and since the Be Informed TBox usually does not change, it works in most cases.

Naïve with heuristic

This verbalizer looks in many ways similar to the completely naive verbalizer, explained on the previous slide. The classes are rebuild to CommonNouns, the ObjectProperties get rewritten to functions and the namedIndividuals to ProperNames. The nice thing though is that it solves the problem on the verbs in a completely different manner. It has an integrated lemmatizer for English, which can retrieve most infinitive forms of verbs automatically. Also is contains an aggregation to concatenate sentences that have the same subject individual and function, so that the verbalizations look more natural.

LemonAided

For the LemonAided verbalizer, An Earley parser is implemented to obtain the lexical information from within each of the labels. Also this verbalizer contains a lemmatizer for both English and Dutch in order to not only rewrite the verbs in the functions, but also to rewrite the verbs that occur inside labels of individuals. All the entities, together with their identifiers, lexical enrichments and lexical adaptations get written into scala entries and get passed to LeMOn (John McRae) and LeMOn2GF (Christina Unger). LeMOn builds a so-called frame around each of the scala entries depending on both the syntactical information. This frame is later used by LeMOn2GF in order to create verbalizations in GF that can deal with label variance.

2.3 Phase 3: Proposed development model

Every grammar generation technique described above, has its own strength and weaknesses. Some methods required a small amount of engineering while others require more. However, the quality of the sentences of some methods should be better than other methods.

When developing the different components, we found that different chains could be implemented depending on what components are used from start to goal, i.e. from Be Informed model to grammar. Therefore we propose an open source development model for language technologies. Figure 5 shows this model, together with the components as described earlier. The model contains an interface for different resources, such as OWL, GF, Be Informed model, Lemon, etc. The other interface in this model is de processor interface. This is the interface that is implemented by the different component as described above. Every component has a from and a to. For example, the naïve method takes a Be Informed model and produces a GF grammar. The resource pool implementation registers different processors, and defines a start resource and a goal resource. From that point it start searching in the list of registered components if there is a component that creates a resource which is equal to the goal resource. If that the start resource is equal to the input resource of the processor, this processor could be used to get from start to goal. If this is not the case, the input resource of that processor becomes the new temporary goal, and the resource pool uses recursion to find a processor that takes as input the start resource and as output the temporary goal resource. This continues until a path is found from start resource to goal resource.

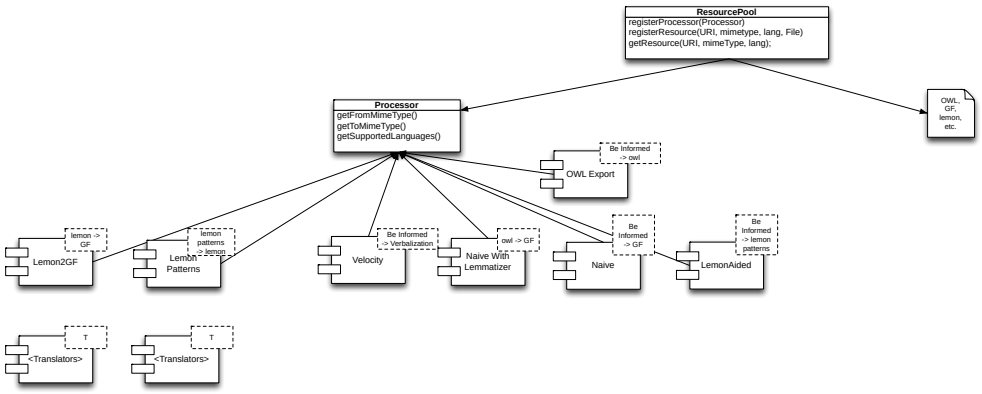

Figure 5. Component model

The whole idea of this model is that everybody could contribute their processor components. For example, new methods that implement another way of lexicalizing ontology labels could be added to the model just by implement it as a processor. A stable version could be created

from the different components to be used for other researchers or businesses.

3 Evaluation of Verbalization Techniques based on GF

3.1 Background and Evaluation Methodology

The different methods proposed earlier, including the baseline method, all generate sentences that are different in quality. To evaluate the quality of the sentences, a language model could be used that calculates the likelihood of the test sentences given a corpus of training sentences. By comparing the likelihood scores of every method, a better understanding of the quality of the different methods could be acquired. Below the evaluation method and the experimental setup are described, and the results are discussed.

3.2 Evaluation Methods

The idea behind a language model is that the probability of a word in a sentence can be determined given all the previous words in the sentence, i.e. P(wn | w1, w2, w3, .., wn-1). This calculation, however, could be simplified by assuming that the *n*th word in a sentence only depends on the previous words (bigram) or the previous two words (trigram), i.e. Markov assumption.

The probability of a sentence can then be calculated by multiplying the probabilities of every bigram/trigram in the sentence. For example, P(w1, w2, w3, w4) = P(w1 | <s>) * P(w2 | w1) * P(w3 | w2) * P(w4 | w3) * (</s> | w4), using a bigram model. Here the <s> indicates the start of a sentence and </s> the end.

The probabilities, such as P(w2 | w1), could be calculated given a corpus of sentences. In this particular case, P(w2 | w1) = P(w1, w2) / P(w1). P(w1, w2) and P(w1) could be calculated by counting the number of occurrences in a corpus.

3.3 Experimental Setup

To be able to perform the evaluation, a Python script was implemented using NLTK (http://nltk.org). NLTK provides a module called Ngram (http://nltk.googlecode.com/svn/trunk/doc/api/nltk.model.ngram.NgramModel-class.html) which could build a Ngram model given a corpus. The eurparl corpus (http://www.statmt.org/europarl/), Dutch-English, was used for all experiments. Where the English part of the corpus was used to evaluate English sentences, and the Dutch part for evaluating Dutch sentences. Also for all experiments a trigram model was trained given the corpus.

Four experiments were conducted using the parameters as described above. Every experiment evaluated for one method each sentences using the trained language model. These sentences were created by using the Housing Benefit model as modelled in Be Informed. An OWL export was also created of this model. The result of each experiment is a scatterplot of the sentence probabilities. The sentences produced by the Velocity method are used as a baseline method, due to the fact that it does not rely on GF.

3.4 Results and Discussion

Figure 7 and 8 show the result for English and Dutch sentences respectively. Figure 7 shows that the spread of the scores for the Velocity verbalizers in English is way more skewed than the spread of the verbalizers making use of GF. Also for the Dutch sentences the GF verbalizers have a more equal spread than the Velocity verbalizers. Notice however that the scores for the naïve with heuristic verbalizer are not taken into account for the Dutch sentences. Since we did not have a lemmatizer for Dutch, we were not able to create sentences in Dutch with this verbalizer.

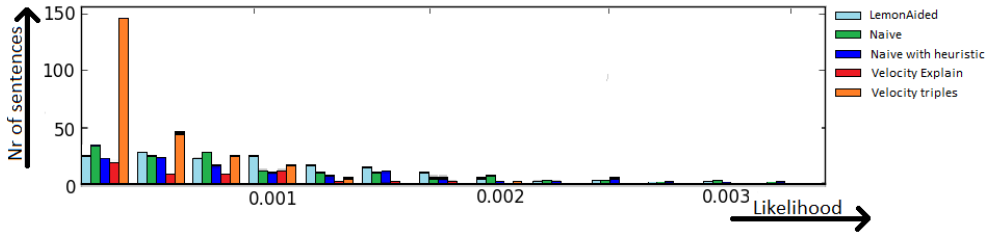

Figure 6. Bar chart showing the likelihood of the sentences for English per verbalizer

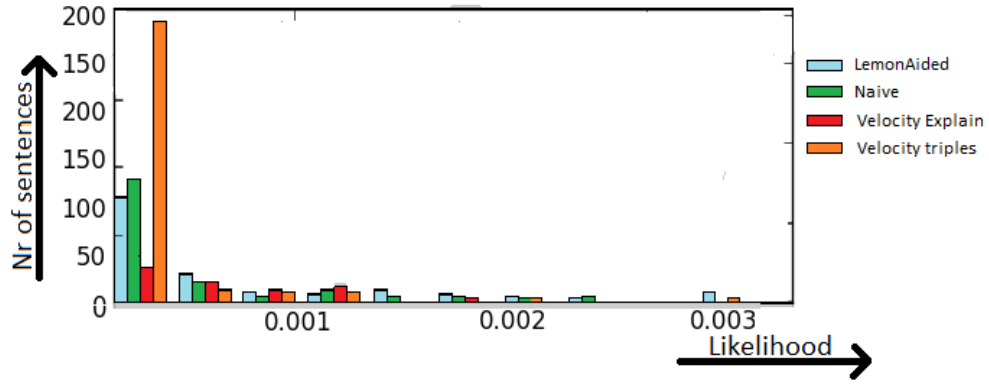

Figure 7. Bar chart showing the likelihood of the sentences in Dutch per verbalizer

The reason for the differences between the two types of verbalizers (Velocity and GF based) is that while GF has a robust way of generating equally correct sentences for much broader variety of sentences than the velocity templates. Together with the fact that the framework we created for GF verbalizations is more reusable, maintainable and manageable than the velocity templates simply makes GF the far more preferable choice.

Wapenrustlaan 11-31
7321 DL Apeldoorn
The Netherlands
T +31 (0)55 368 14 20
E
info@beinformed.com
www.beinformed.com

About Be Informed

Be Informed is an internationally operating, independent software vendor. The Be Informed business process platform supports administrative processes, which are becoming increasingly knowledge-intensive. Thanks to Be Informed's unique approach to dynamic case management, the next wave after business process management, organizations using Be Informed often report cost savings of tens of percents. Further benefits include a much higher straight-through processing rate leading to vastly improved productivity, and a reduction in time-to-change from months to days.