D4.3 GRAMMAR-ONTOLOGY INTEROPERABILITY

HTTP://WWW.MOLTO-PROJECT.EU

Contract No.:	FP7-ICT-247914
Project full title:	MOLTO - Multilingual Online Translation
Deliverable:	D4.3 Grammar-Ontology Interoperability
Security (distribution level):	Public
Contractual date of delivery:	May 2012
Actual date of delivery:	May 2012
Type:	Prototype
Status & version:	Final
Author(s):	Milen Chechev, Aarne Ranta, Mariana Damova, Ramona Enache
Task responsible:	ONTO (WP4)
Other contributors:	

ABSTRACT

This document describes the interoperability between the Grammatical Framework (GF) grammar and a semantic repository, which aims to bridge the gap between natural language (NL) and formal knowledge. Two different approaches are presented: semi-automatic with manual intervention and template-based GF grammar generation.

We describe in detail the exploitation of the prototypes and tools that use these approaches.

CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

The structured knowledge on the global network is constantly increasing. This can be seen with the constantly increasing size of the LOD cloud and the initiatives of the biggest players on the market - Google Knowledge Graph and Facebook Open Graph. However, there is still a gap between structured semantic knowledge and natural language. Traditionally, SPARQL query language is used for querying the semantic data and the results are represented as a set of triples. But in fact common users always prefer natural language for their queries.

Both the Knowledge Representation Infrastructure (KRI) described in Deliverable 4.1 (1) and the data models in Deliverable 4.2 (2) are used for building a prototype and tools that address the above issues. The prototype, presented in this deliverable, interprets natural language questions in different languages, searches for data in a semantic repository and presents the results in the requested language. This is possible because of the GF framework[1] and the interoperability between the GF grammar and the ontologies.

The prototype, developed for this deliverable, works with controlled natural language. Controlled language is a natural language with restricted grammar and vocabulary. These restrictions aim to eliminate language ambiguity and complexity. We use GF as a framework for processing the language. The abstract grammar covers all sentences and the concrete grammars deal with the spoken languages - English, French, German, Italian, Swedish and Finnish. Additional concrete grammars could be added in the future.

The interoperability between natural language and ontologies will provide users with an easy interface for querying and retrieving semantic data. The main novelty of our approach is that the user will be able to make queries in all languages covered by the GF *abstract representation*, which is independent from the specific language. Then, the abstract representation will be converted to a SPARQL query, which will be executed against the semantic repository. The results obtained from the semantic search will be transformed to a GF abstract representation, from which an answer in natural language will be generated.

[1] http://gramaticalframework.org

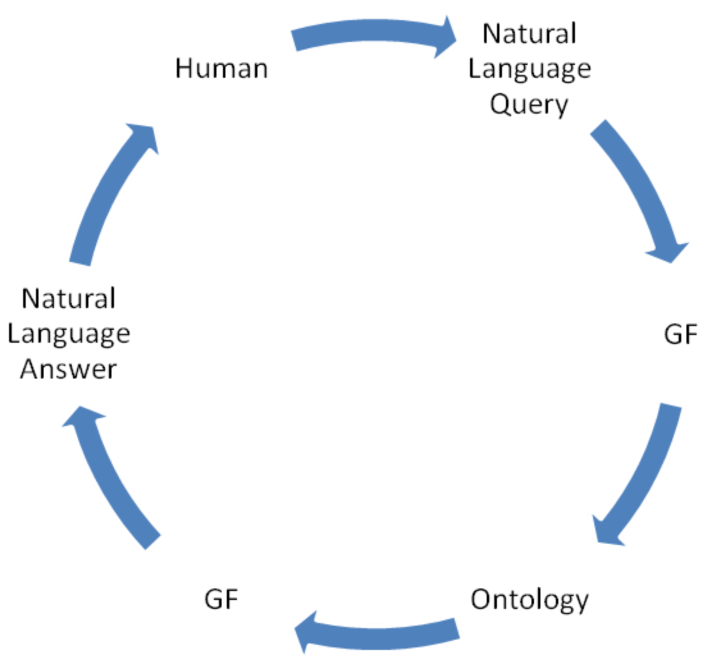


Figure 1. Question answering circle

2 NATURAL LANGUAGE TO SEMANTIC REPOSITORY

Natural language is the normal human way of making a query but to be able to apply it to a semantic repository it has to be mapped to a formal query language. We restrict the queries to a controlled natural language to eliminate the ambiguity. Depending on the available resources, there are several approaches to the interoperability between the controlled language and the semantic repository. Here we explore the two most common situations:

- We already have the controlled language and the semantic repository, and we want to connect them.
- We have a semantic repository and we have to build a controlled query language to cover the classes and instances in the repository.

2.1 GENERIC GF QUERY GRAMMAR

A generic GF query grammar was built as part of the MOLTO project. It can be downloaded from: http://www.grammaticalframework.org/examples/query/small. The grammar covers general knowledge questions such as “show me all people/organizations/locations”, “show me people/organizations/locations located in somewhere”, etc. It also describes the general knowledge that can be used and explored in every domain. It can also be used as a basis for a domain specific query grammar, which an experienced GF user could write in a short time.

Once the GF query grammar is created, it has to be mapped to the SPARQL syntax. There are two possibilities here - to make a concrete SPARQL syntax of the GF grammar or to make mapping rules between the GF abstract syntax and the SPARQL syntax. The first one is demonstrated as part of the next section - the SPARQL concrete grammar is automatically built and used for transforming queries from natural language to SPARQL. This section focuses on the second approach – the semi-automated mapping between the GF abstract syntax and SPARQL. It is more generic and easier to maintain when more sophisticated SPARQL queries need to be executed.

2.1.1 MAPPING-RULES TOOL

The mapping-rules tool provides a semi-automatic transformation from the GF Abstract Representation to SPARQL. The transformation is realized by rules written in the language of the tool.

There are several constructions in the language, which ensure the mapping functionality.

The first one is the “define clauses”:

```
#define nameOfDefine() { SELECT }
```

This construction demonstrates the usage of **`nameOfDefine()`**. In the example, all occurrences of **`nameOfDefine`** below the “define clause” will be replaced with the text “SELECT”. This usage of define is similar to the one in the c/c++ languages.

```
#define nameOfDefine() { SELECT ## " " ##DISTINCT }
```

This defines that all occurrences of **`nameOFDefine()`** will be substituted with "SELECT DISTINCT". The symbol ## is used to concatenate strings (this symbol will be used in all rules below with the same meaning).

The next construction is the *table* definition, which maps a set of words to a set of other words:

```
#table nameOfTable[2] {
  Person       <http://proton.semanticweb.org/protontop#Person>;
  Location     <http://proton.semanticweb.org/protontop#Location>;
  Organization <http://proton.semanticweb.org/protontop#Organization>;
}
```

This construction defines a mapping table named **`nameOfTable`**. It can be used in the rules for reducing the duplication of long literals and for easier writing of different constants, such as URIs, names, etc.

The syntax of the language provides an easy way to write comments:

```
//comment - this is the construction for comment
```

The main part of the mapping language consists of the rules. They have three parts.

Example:

```
(QSet ?X) | single(X) && type(X) == "" --> construct WHERE { sparqlVar(name(X)) rdftype() class(name(X)) . sparqlVar(name(X)) rdfslabel() sparqlVar(name(X)) ## "_label". sparqlVar(name(X)) ?p ?o};
```

The first part is used as a regex that tries to match the GF Abstract Representation. The end of the first part and the beginning of the second part is marked with the symbol "|". The second part is used as a Boolean condition for executing the rule. In the example, "?X" is bound with the rest of the Abstract Representation after the *QSet* word. The function *single* is used to determine if X is a tree or a single term. The function *type* can return a blank string or a Person, Organization, Location, JobTitle, etc. The third part of the rule determines the SPARQL query that is matched. In the example, *sparqlVar* is a table that has to be defined at the beginning of the file and maps the name of the variable X to a SPARQL variable. *rdfslabel* is also defined at the beginning of the file. The string ## is used to denote the concatenation of the name of the SPARQL variable and *"_label"*. This is very useful for dynamic creation of names.

The next section describes the Query Grammar, which was mapped with the mapping-rules tool to the semantic repository. The setting for the MOLTO project uses the OWLIM semantic repository [11].

2.1.2 QUERY GRAMMAR

The query grammar is built[2] as a generic query grammar that covers questions about people, organizations, locations and job titles. It is general enough to be used with different semantic repositories that contain general knowledge data and it can be further extended to cover more specific queries.

The GF Abstract grammar includes 15 categories and 33 functions. To keep it more generic, the semantic data is not included in it. Below is an example of how such data can be included:

```
Person1 : Pers ;
Organization1 : Org ;
Location1 : Loc ;
JobTitle1 : JobTitle ;
```

The idea is that the specific lexicon data is extracted from the semantic repository and then automatically added to the grammar.

[2] http://www.grammaticalframework.org/examples/query/small

If the administrator needs to make a custom controlled language, which does not only have a custom lexicon, but also custom functions, he will have to make changes to the GF grammars.

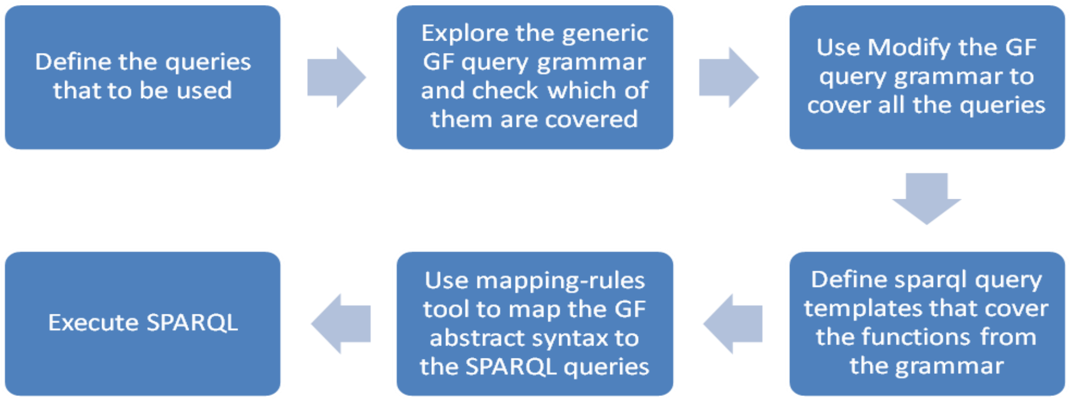
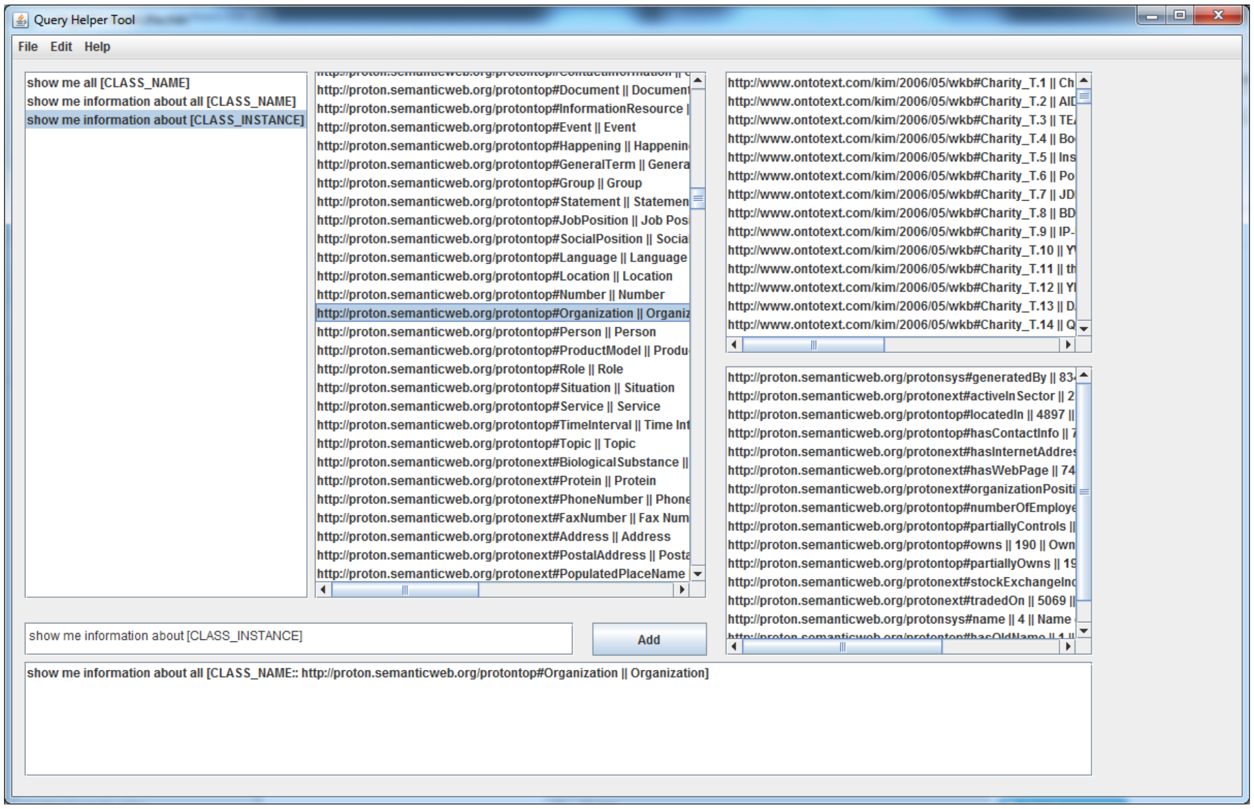

Define the queries that to be used
Explore the generic GF query grammar and check which of them are covered
Use Modify the GF query grammar to cover all the queries
Define sparql query templates that cover the functions from the grammar
Use mapping-rules tool to map the GF abstract syntax to the SPARQL queries
Execute SPARQL

Figure 2. The Generic Query grammar usage process

2.2 AUTOMATICALLY BUILT QUERY GRAMMAR

There are situations in which we have semantic data and want to provide a natural language interface for it. For example, if the user does not have much experience with natural language processing or the GF, but he wants to use some tools for GF generation. In that case he needs to have a tool for generating a controlled language, which would provide the connection between the natural language and SPARQL. Besides, if the user's ontology is multilingual, he would also expect to be able to make questions in all data languages.

The GF query helper builder (GQHB) tool is designed exactly for such situations. It provides all necessary functionalities – a connection to the SPARQL endpoint, extraction of specific data from the semantic repository, support for the user when selecting which parts of the data to be queried, and finally, generation of a controlled language by using the templates and the semantic data. The user interface of the tool is shown on Figure 3.



```
show me all [CLASS_NAME]
show me information about all [CLASS_NAME]
show me information about [CLASS_INSTANCE]
```

```
http://proton.semanticweb.org/protontop#Document || Document
http://proton.semanticweb.org/protontop#InformationResource |
http://proton.semanticweb.org/protontop#Event || Event
http://proton.semanticweb.org/protontop#Happening || Happenin
http://proton.semanticweb.org/protontop#GeneralTerm || Genera
http://proton.semanticweb.org/protontop#Group || Group
http://proton.semanticweb.org/protontop#Statement || Statemen
http://proton.semanticweb.org/protontop#JobPosition || Job Pos
http://proton.semanticweb.org/protontop#SocialPosition || Social
http://proton.semanticweb.org/protontop#Language || Language
http://proton.semanticweb.org/protontop#Location || Location
http://proton.semanticweb.org/protontop#Number || Number
http://proton.semanticweb.org/protontop#Organization || Organiz
http://proton.semanticweb.org/protontop#Person || Person
http://proton.semanticweb.org/protontop#ProductModel || Produ
http://proton.semanticweb.org/protontop#Role || Role
http://proton.semanticweb.org/protontop#Situation || Situation
http://proton.semanticweb.org/protontop#Service || Service
http://proton.semanticweb.org/protontop#TimeInterval || Time Int
http://proton.semanticweb.org/protontop#Topic || Topic
http://proton.semanticweb.org/protonext#BiologicalSubstance ||
http://proton.semanticweb.org/protonext#Protein || Protein
http://proton.semanticweb.org/protonext#PhoneNumber || Phone
http://proton.semanticweb.org/protonext#FaxNumber || Fax Num
http://proton.semanticweb.org/protonext#Address || Address
http://proton.semanticweb.org/protonext#PostalAddress || Posta
http://proton.semanticweb.org/protonext#PopulatedPlaceName
```

```
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.1 || Ch
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.2 || AID
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.3 || TE
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.4 || Bo
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.5 || Ins
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.6 || Po
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.7 || JD
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.8 || BD
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.9 || IP-
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.10 || Y
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.11 || th
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.12 || Y
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.13 || D
http://www.ontotext.com/kim/2006/05/wkb#Charity_T.14 || Q
```

```
http://proton.semanticweb.org/protonsys#generatedBy || 83
http://proton.semanticweb.org/protonext#activeInSector || 2
http://proton.semanticweb.org/protontop#locatedIn || 4897 ||
http://proton.semanticweb.org/protontop#hasContactInfo || 7
http://proton.semanticweb.org/protonext#hasInternetAddres
http://proton.semanticweb.org/protonext#hasWebPage || 74
http://proton.semanticweb.org/protonext#organizationPositi
http://proton.semanticweb.org/protontop#numberOfEmploye
http://proton.semanticweb.org/protontop#partiallyControls ||
http://proton.semanticweb.org/protontop#owns || 190 || Own
http://proton.semanticweb.org/protontop#partiallyOwns || 19
http://proton.semanticweb.org/protonext#stockExchangeInd
http://proton.semanticweb.org/protonext#tradedOn || 5069 ||
http://proton.semanticweb.org/protonsys#name || 4 || Name
```

```
show me information about [CLASS_INSTANCE]
```

```
show me information about all [CLASS_NAME:: http://proton.semanticweb.org/protontop#Organization || Organization]
```

Figure 3. GQHB tool interface

The GQHB tool uses the text data trom the different classes and instances and combines it with grammatical templates to provide a controlled language for a specific domain.

The main components of the tool are the query templates. The user views them as natural language sentences with some bindings. The current possible bindings are [CLASS_NAME], [CLASS_INSTANCE], and [PREDICATE].

The [CLASS_NAME] bindings can be bound to a specific class. After that, the label of the class is used as a resource for the natural language question. For example, we select the template "`show me more information about all [CLASS_NAME]`", and bind <http://proton.semanticweb.org/protontop#Organization> to the [CLASS_NAME]. The label of the class will be used and the query will be equal to "`show me more information about all organizations`".

```
MQuery  : Query -> Move ;
QSet    : Set  -> Query ;
SAll   : Kind -> Set ;
QInfo   : Set  -> Query ;
[CLASS_NAME] : Kind ;
```

Figure 4. Abstract grammar section from GQHB tool's configuration

Next, the template defines abstract syntax and concrete syntax sections that need to be included in the abstract and concrete grammars, which are going to be generated by the tool. The abstract syntax section is shown on Figure 4, the concrete English grammar section - on Figure 5 and the concrete French grammar section - on Figure 6.

```
MQuery  q = q ;
QSet s =
      let
       ss : NP = s
            | mkNP (mkNP thePl_Det (mkN "name")) (mkAdv possess_Prep s)
            ---- s's names
      in
        mkUtt (mkImp (mkVP (mkV3 give_V) (mkNP i_Pron) ss))
      | mkUtt (mkQS (mkQCl (L.CompIP whatSg_IP) ss))
      | mkUtt (mkQS (mkQCl (L.CompIP (L.IdetIP (mkIDet which_IQuant)))
ss))
      | mkUtt ss ;
QInfo  s =
      let
        info : NP = mkNP (mkNP (mkN "information")) (mkAdv (mkPrep
"about") s)
      in
        mkUtt (mkImp (mkVP (mkV3 give_V) (mkNP i_Pron) info))
      | mkUtt info ;
SAll k = mkNP all_Predet (mkNP aPl_Det k) | mkNP thePl_Det k ;
```

Figure 5.Concrete English grammar section from GQHB tool's configuration

```
MQuery  q = q ;
QSet s = mkUtt (mkVP (mkV2 give_V) s) ;
QInfo s = mkUtt (mkVP (mkV2 give_V)
              (mkNP  all_Predet  (mkNP  thePl_Det  (mkCN  information_N
(mkAdv on_Prep s))))) ;
SAll k = mkNP all_Predet (mkNP thePl_Det k) ;
```

Figure 6.Concrete French grammar section from GQHB tool's configuration

There is also a SPARQL concrete grammar that is generated from the GQHB tool. The SPARQL concrete grammar section for the example above is presented on Figure 7.

```
            MQuery  q = q;
            QSet  k = {s = "construct WHERE {" ++ k.s ++ "}"};
            SAll  k  =  {s  =  "?subject  &lt;http://www.w3.org/1999/02/22-
rdf-syntax-ns#type&gt;" ++ k.s ++"."};
            QInfo  k  =  {s  =  "construct  WHERE  {"  ++  k.s  ++  "?subject
?predicate ?object.}"};
```

Figure 7.Concrete SPARQL grammar section from GQHB tool's configuration

All these segments are included in the templates file, which is loaded as a configuration file of the GQHB tool, and they are used to generate a parallel multilingual query grammar for the semantic repository. The main goal is to make the templates general enough so they can be used in different domains. Additionally, when using the bindings the variety of the query language becomes relatively large.

The process of generating GF grammars with the GQHB tool is shown on Figure 8.

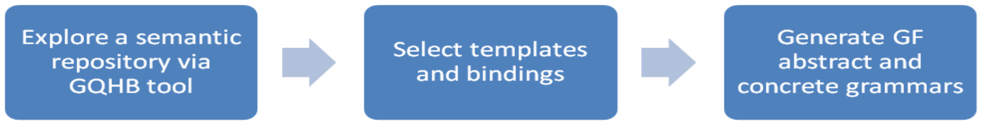

Explore a semantic repository via GQHB tool
Select templates and bindings
Generate GF abstract and concrete grammars

Figure 8. GQHB tool process stream

3 RDF RESULTS TO NATURAL LANGUAGE

The ontology verbalization to a controlled language is a research topic that was continuously revisited in the last few years. Several systems have been reported, such as ACE[6], Sydney Sintax[7], Text Generation from Ontologies[8], etc. but none of them is multilingual. To support multilingualism we use a GF for translating the results and then we need to map them to a GF Abstract Grammar that covers the semantic data.

Several papers have been submitted that present ontology verbalization with GF.

[10] is about the verbalization of the SUMO ontology using GF, and [4] describes the verbalization of the ontology from the cultural heritage domain. The next sections summarize the possible approaches. Section 3.1 represents a fully automatic approach that can have a large coverage, while Section 3.2 - the approach with manually making discursive patterns in the GF, which then are used for better verbalization.

3.1 AUTO GENERATE ANSWERS GRAMMAR FROM SEMANTIC REPOSITORY

The natural-language-answers tool has been implemented as part of the work on verbalizing the SPARQL query results. The tool provides:

1. functionality for transforming a list of ontology files, presented in turtle format to the GF abstract and concrete grammars
2. real-time transformation of a list of triples to natural language using the GF abstract and concrete grammars, built in the previous step

The automatic generation of the GF grammar is tested against a world knowledge base ontology that contains information about people, locations and organizations. The automatic generation is based on the ontology structure, instances and their properties. The ontology classes and labels are used to provide information about the instance type. The predicates are used to provide the grammatical rules about the relations between the objects.

The technology of the automatic creation of the GF grammar is based on the idea of abstract representation. Abstract representation is a language independent representation of the semantics of the sentence, whereas ontology is a formal representation of the semantics. Still, the predicates in the ontology can be expressed as functions of the GF abstract representation syntax, ontology classes can be presented as categories, and instances - as functions of the generated class category.

The main difficulty of this approach is that the GF functions names and their variables are unique. So, in order to prevent the generation of more statements from the ontology, the function name has to be produced from the name of the predicate and the name of the instance classes. For instance, many different classes use the predicate *locatedIn* and all of them have to generate their own functions. An example of an automatically generated abstract grammar is shown on Figure 9.

If we want to present the generated text to the user, we have to define the priority of the predicates and to combine the simple sentences into more complex ones. However, because of some ambiguity we have observed, we selected the discursive patterns as a better approach for generating more complex sentences.

```
abstract Wkbx = {
flags startcat = Phrase;

cat
 Phrase; Bank; Continent; City; University;
Fun
InfoBank : Bank ->Phrase;
InfoContinent : Continent ->Phrase;
InfoCity : City ->Phrase;
InfoUniversity : University ->Phrase;

Bank_T_147 : Bank;
Bank_T_148 : Bank;
Continent_T_1 : Continent;
Continent_T_2 : Continent;
City_T_1 : City;
University_T_1 : University;
locatedInBankCity : Bank  -> City -> Phrase ;
locatedInUniversityCity : University  -> City -> Phrase ;
}
```

Figure 9. Automatically generated abstract grammar

The concrete GF grammar is generated from the text information and the labels connected to the ontology, although it needs some prepossessing to fix the labels that are shortened, misleading or not descriptive enough. An example of such automatically generated concrete English grammar is shown on Figure 10.

Once the grammars are generated, the tool is connected to the semantic repository and the natural language results are retrieved from the queries.

After generating the concrete English grammar, it can be applied to other languages in two steps - by first changing the grammar to use the Resource Grammar Library[10] for the new language and then by using lexicons to translate the words from English to the new language.

```
concrete WkbEng of Wkb =
open MorphoEng, ResEng, ParadigmsEng, MakeStructuralEng, SyntaxEng in
{

lincat Phrase = Cl;
 Bank = NP;
 Continent= NP;
 City= NP;
 University = NP;

lin Bank_T_1 = mkNP( mkN "Bank DSK");
 Bank_T_2 = mkNP( mkN "First International Bank");
 Continent_T_1 = mkNP( mkN "Europe");
 Continent_T_2 = mkNP( mkN "Asia");
 City_T_1 = mkNP( mkN "Sofia");
 University_T_1 = mkNP( mkN "MIT");

InfoBank x = mkCl x (mkN "bank");
InfoCity x = mkCl x (mkN "city");
InfoContinent x = mkCl x (mkN "continent");
InfoUniversity x = mkCl x (mkN "university");
locatedInBankCity x y = mkCl x  (mkVP (passiveVP (mkV2  (mkV "locate")
)) (mkAdv (mkPrep "in") y));
locatedInUniversityCity x y = mkCl x  (mkVP (passiveVP (mkV2  (mkV
"locate") )) (mkAdv (mkPrep "in") y));


}
```

Figure 10. Automatically generated concrete English grammar

3.2 USING DISCURSIVE PATTERNS

The verbalization of the ontology in the cultural heritage domain was shown as part of WP8. It uses discursive patterns that were created especially for the needs of the domain. The main idea is that the structure of the text is preset as a pattern and the different data provides the changes of the information in it. Examples of the text generated with the respective discursive patterns in English, Finnish, French, Italian and Swedish are given below:

- *PaintingEng: The girl was painted on canvas by Anna Lindskog in 1885. It is of size 435 by 365 and it is painted in black. This oil painting is displayed at the City Museum of Gothenburg.*

- *PaintingFin: Maalauksen Flickan on maalannut Anna Lindskog kankaalle vuonna 1885. Se on kokoa 435 kertaa 365 ja se on maalattu mustalla. Tämä öljymaalaus on esillä Göteborgin kaupunginmuseossa.*

- *PaintingFre: Le tableau Flickan a été peint sur toile par Anna Lindskog en 1885. Il est de taille 435 sur 365 et il est peint en noir. Cette peinture à l' huile est exposée dans le musée municipal de Göteborg.*

- *PaintingIta: Il quadro Flickan è stato dipinto su tela da Anna Lindskog nel 1885. Misura 435 per 365 ed è dipinto in nero. Questo dipinto ad olio è esposto nel museo municipale di Goteburgo.*

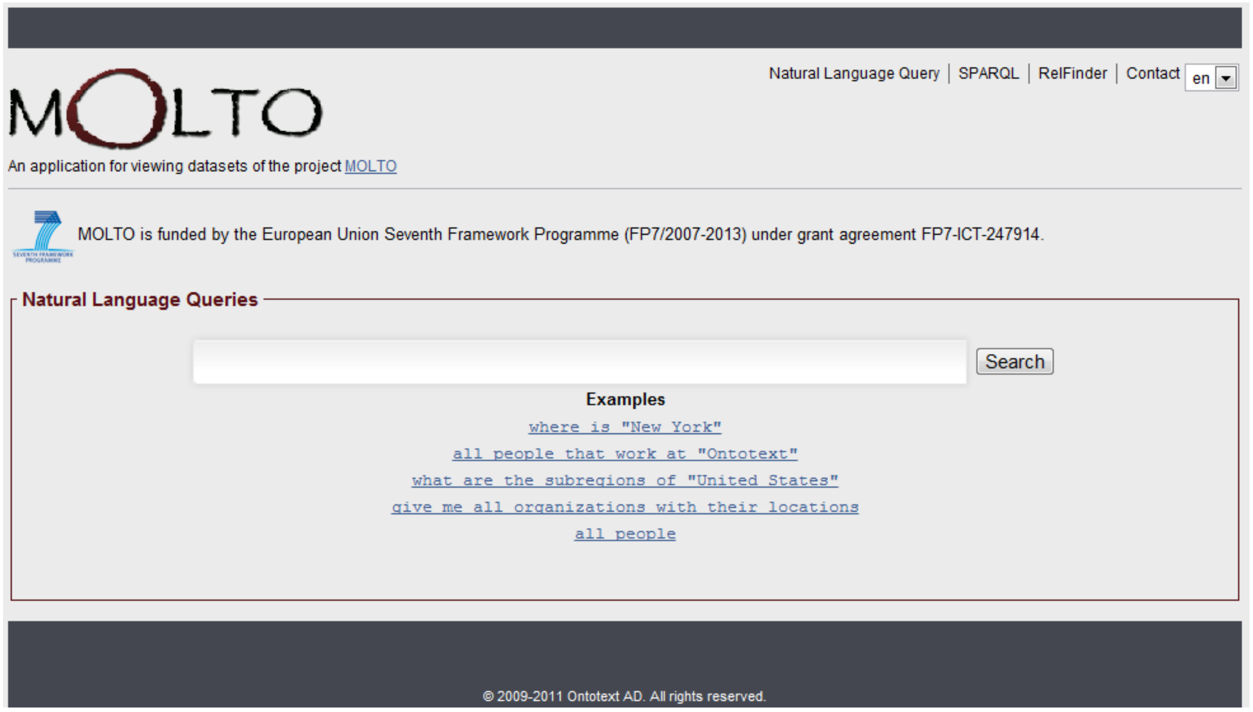
- *PaintingSwe: Flickan målades på duk av Anna Lindskog år 1885. Den är av storlek 435 gånger 365 och den är målad i svart. Den här oljemålningen är utställd på Göteborgs stadsmuseum.*

Using this approach instead of the approach shown in Section 3.1 provides better verbalization. The sentences are more complex, the order of the sentences is fixed but it requires more manual work. The user who generates the discursive patterns has to be experienced both in GF and ontologies, i.e. to be able to analyze the knowledge in the ontology and to provide good representation in GF.

More details about the discursive pattern grammar that was generated for the Goteborg City museum can be found in [5].

4 PROTOTYPE

This chapter describes the system prototype[3] that is built as part of Deliverable 4.3. It is an information retrieval system that retrieves semantic data from a semantic repository. The data loaded in the repository is from the world knowledge base domain and includes data for people, organizations, locations and job positions.



An application for viewing datasets of the project MOLTO

MOLTO is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement FP7-ICT-247914.

where is "New York"
all people that work at "Ontotext"
what are the subregions of "United States"
give me all organizations with their locations
all people

© 2009-2011 Ontotext AD. All rights reserved.

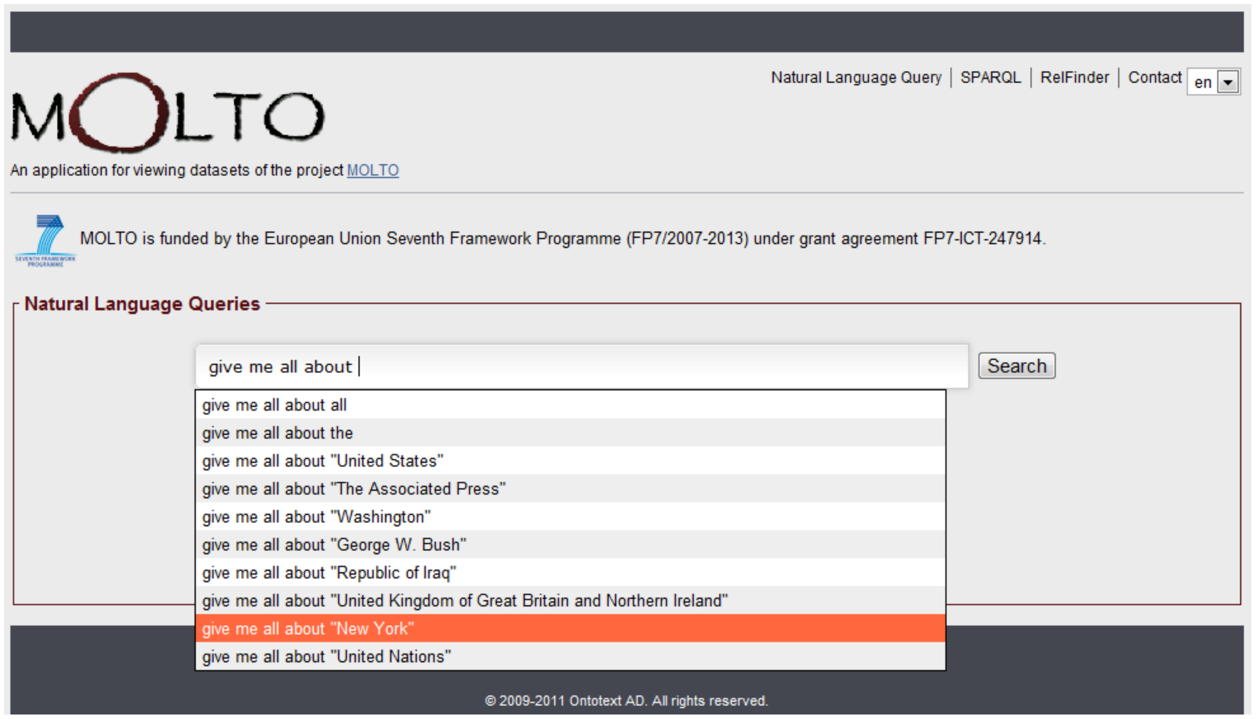
Figure 11. Main page of the prototype

The home page offers the possibility of natural language search with the help of the system's controlled language. You can see the user interface on Figure 11. In the upper right corner there is a drop down menu that can be used to change the language of the interface and the query. There are six languages that are already imported in the prototype. They are:

- English
- German
- French
- Finnish
- Swedish
- Italian

The autocomplete function is available in all of the above written languages. Figure 12 shows an example of the autocomplete function.

[3] http://molto.ontotext.com


Natural Language Query | SPARQL | RelFinder | Contact en

An application for viewing datasets of the project MOLTO

MOLTO is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement FP7-ICT-247914.

Natural Language Queries

give me all about

give me all about all
give me all about the
give me all about "United States"
give me all about "The Associated Press"
give me all about "Washington"
give me all about "George W. Bush"
give me all about "Republic of Iraq"
give me all about "United Kingdom of Great Britain and Northern Ireland"
give me all about "New York"
give me all about "United Nations"

© 2009-2011 Ontotext AD. All rights reserved.

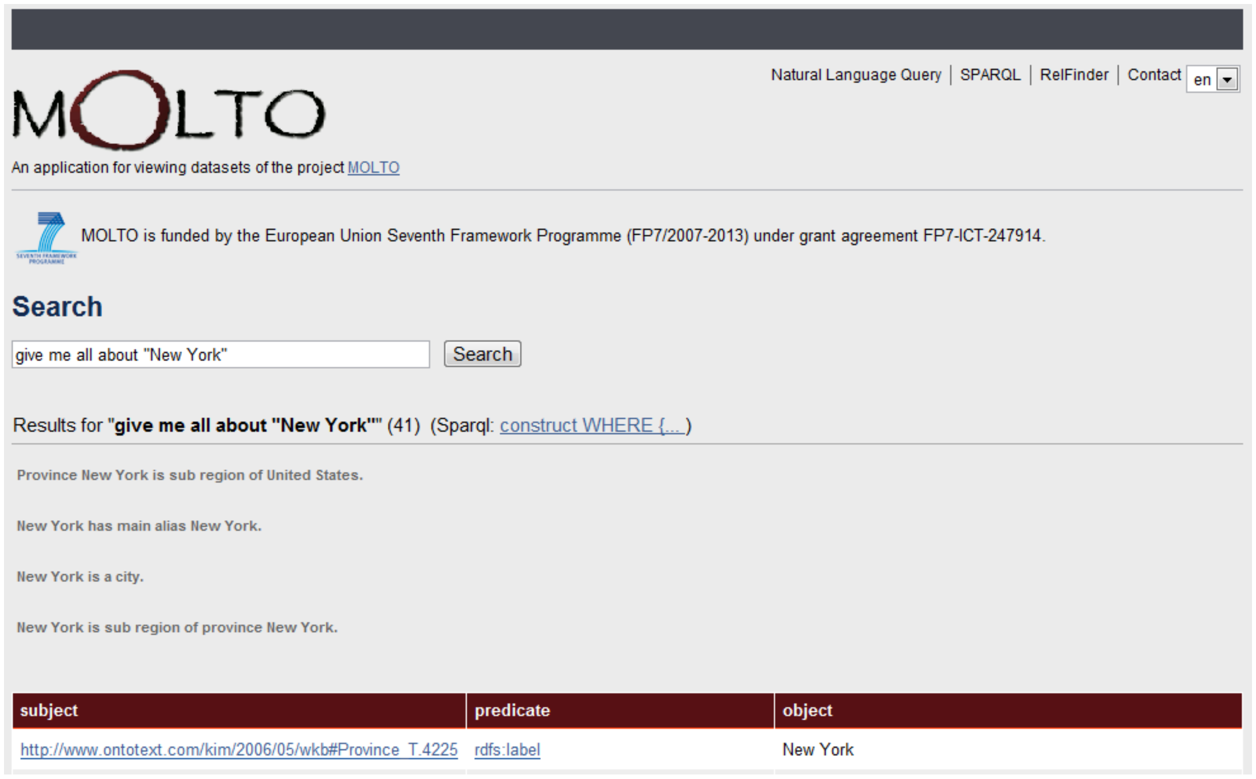
Figure 12. The autocomplete example.

After the users create their query, they can execute it against the OWLIM[4] semantic repository. The results are shown on Figure 13. They are presented as several sentences in natural language, followed by a table with semantic results returned from the semantic repository. Currently only English is covered as a language for the returned results.

The current representation of the natural language query "Show me all about New York" is:

```
construct WHERE {
?location <http://www.w3.org/2000/01/rdf-schema#label> "New York" .
?location                                   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://proton.semanticweb.org/protontop#Location> .
?location ?p ?o.
}
```

[4] http://www.ontotext.com/owlim



An application for viewing datasets of the project MOLTO

MOLTO is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement FP7-ICT-247914.

Search

give me all about "New York" Search

Results for "**give me all about "New York"**" (41) (Sparql: construct WHERE {...})

Province New York is sub region of United States.

New York has main alias New York.

New York is a city.

New York is sub region of province New York.

subject	predicate	object
http://www.ontotext.com/kim/2006/05/wkb#Province_T.4225	rdfs:label	New York

Figure 13. Example for search results

5 DEVELOPERS GUIDE

All described tools can be downloaded from the molto svn server: svn://molto-project.eu/wp4

As prerequisites to build them there is a need of maven and jdk 1.6.

The projects have the following structure:

- molto-core
  - fsa - a module that implements a final state automat that is used for auto-complete
  - lexicons - a module that implements a final state automat for the lexicon words
  - mapping-rules - a module that connect the GF abstract representation to sparql queries.
  - molto-repository-helper - a GUI tool that can be connected to a sparql endpoint and to generate a GF grammar based on predefined template file.
  - natural-language-answers - a module that verbalize results from a sparql query.
  - natural-language-queries - a module that depends on fsa, lexicon and mapping rules and provide auto-complete for natural language queries and transformation to sparql.
- molto-web - the web interface to a semantic repository that use the modules above.
- molto-kri - customization of the web interface that is used for http://molto.ontotext.com

More details about building the projects can be found at the folders of the projects at readme.txt file.

6 EXTENDING THE SCOPE

The prototype can be extended by:

- adding more databases
- extending the queries
- extending the answers
- adding more languages

If we want to add more data to the semantic repository we have to use one of the interfaces for the OWLIM semantic repository (e.g. the sesame openrdf tools). Once added, the data will be available for searching and if the scope of the data is the same as the already implemented queries, it will be retrieved as a result.

If we want to extend the scope of the queries, the GF grammars have to be changed to cover the new sentences. Next, the mapping rules to SPARQL have to be provided.

Currently the GF grammar used for answering is automatically built from the ontology. Consequently its quality suffers, but it can be manually manipulated to give better results. Another scenario in which we may need to manipulate the GF grammar for the results is when new databases are added or if we need to cover a new language.

To add a new language for natural language queries to the prototype, we need a concrete GF grammar. It can be built by using some of the already existing grammars as an example and manually doing modifications such as changing the lexicon and fixing more specific constructions.

7 CONCLUSIONS

In this document we presented the tools and the prototype providing interoperability between the grammars and the ontologies. The prototype shows the practical use of this interoperability - the natural language interface to a semantic repository and the verbalization of results from the queries against it. The explored approaches were demonstrated with the implementation of specific tools, but some of them still required manual work, especially the queries to more specific domains or verbalizing the results with more sophisticated sentences. Apart from that, we have observed that the implemented functionalities for querying the semantic repository, using natural language and retrieving results again in natural language, have made the system more accessible to the users who are not experts in the semantic repository domain.

8 REFERENCES

1. **Mitankin, P. and Ilchev, A.** *D4.1 Knowledge Representation Infrastructure.MOLTO deliverable 4.1.* November 2010.
2. **Damova, M.** *D4.2. Data Models and alignment. MOLTO Deliverable 4.2.* May 2011.
3. **Dannells, D., Damova, M. Enache R., Chechev, M**. An Infrastructure for Integrating and Accessing Museum Linked Data In: Proceedings of Workshop Language Technologies for Digital Humanities and Cultural Heritage, RANLP'2011, Hissar, Bulgaria, September 2011.
4. **Dannells, D., Enache E., Damova M., Chechev M**. Multilingual Online Generation from Semantic Web Ontologies. In: Proceedings of WWW'2012, Lyon France, April 2012
5**. Dannélls, D., Ranta A., and Enache R**. D8.2 Multilingual grammar for museum object descriptions
6. **Kaljurand, K**. "ACE View – an ontology and rule editor based on Attempto Controlled English" Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with ISWC 2008
7**. Cregan, A., Schwitter, R., Meyer, T**.: Sydney OWL Syntax - towards a Controlled Natural Language Syntax for OWL 1.1., Proceedings of the Fourth OWLED Workshop on OWL Experiences and Directions, 2007.
8. **Bontcheva, K.** "Generating Tailored Textual Summaries from Ontologies", Proceedings of the European Semantic Web Conference, 2005
9. **Enache, R. and Angelov, K**. Typeful ontologies with direct multilingual verbalization.Workshop on Controlled Natural Languages (CNL), 2010.
10. **Ranta, A**. The GF resource grammar library. Linguistic Issues in Language Technology, 2(2), 2009.
11. **Bishop, B., Kiryakov, A., Ognyanov, A., Peikov, I., Tashev, Z., and Velkov, R**. OWLIM: A family of scalable semantic repositories. Semantic Web Journal, Special Issue: Real-World Applications of OWL, 2011.