



## D4.3 GRAMMAR-ONTOLOGY INTEROPERABILITY

*[HTTP://WWW.MOLTO-PROJECT.EU](http://www.molto-project.eu)*

<b>Contract No.:</b>	FP7-ICT-247914
<b>Project full title:</b>	MOLTO - Multilingual Online Translation
<b>Deliverable:</b>	D4.3 Grammar-Ontology Interoperability
<b>Security (distribution level):</b>	Public
<b>Contractual date of delivery:</b>	30.09.2010
<b>Actual date of delivery:</b>	30.09.2010
<b>Type:</b>	Prototype
<b>Status &amp; version:</b>	Final
<b>Author(s):</b>	Milen Chechev
<b>Task responsible:</b>	ONTO ( <a href="#">WP4</a> )
<b>Other contributors:</b>	

### ABSTRACT

THIS DOCUMENT DESCRIBES THE MOLTO D4.3 PROTOTYPE HOSTED ON [HTTP://MOLTO.ONTOTEXT.COM](http://molto.ontotext.com), GIVE MORE DETAILS ABOUT THE DIFFERENT MODULES THAT THE PROTOTYPE CONTAINS AND INSTRUCTIONS ABOUT EXPLOITATION.

## CONTENTS

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
<b>2</b>	<b>The Architecture.....</b>	<b>3</b>
<b>3</b>	<b>Auto complete tools .....</b>	<b>4</b>
<b>4</b>	<b>Mapping rules .....</b>	<b>5</b>
<b>5</b>	<b>Results to Natural Language.....</b>	<b>5</b>
<b>6</b>	<b>Prototype.....</b>	<b>7</b>
<b>7</b>	<b>Extending THE SCOPE .....</b>	<b>9</b>
<b>8</b>	<b>Conclusions.....</b>	<b>10</b>
<b>9</b>	<b>Future work .....</b>	<b>10</b>
<b>10</b>	<b>References.....</b>	<b>10</b>
<b>11</b>	<b>Appendix A - Mapping rules syntax.....</b>	<b>11</b>

## TABLE OF FIGURES

Figure 1. Question answering circle.....	4
Figure 2. Automatic generated abstract grammar.....	6
Figure 3. Automatic generated concrete English grammar.....	7
Figure 4. Main page of the prototype.....	7
Figure 5. The auto complete example.....	8
Figure 6. Example for results from search.....	9

# 1 INTRODUCTION

In everyday life we often look for answers of different questions and the most natural human way of representing the question is with use of our natural language. The main problems when a program try to retrieve answer of these kind of questions are: difficulties with understanding natural language question, difficulties with processing more than one natural language, difficulties with translating the result of the language of the user.

The knowledge representation infrastructure described at Deliverable 4.1 (1) and data models described at Deliverable 4.2 (2) are both used to build a prototype that face the above problems. The prototype can process natural language question on different languages, search for data in semantic repository and retrieve the data in the requested language. All this is possible because of the use of GF framework<sup>1</sup> and the interoperability between the GF grammar and the ontologies.

The prototype can work with natural language queries but have and some restrictions. The main restriction that is made is the use of the controlled language. The controlled language is natural language with restricted grammar and vocabulary. This restriction is made to eliminate ambiguity and complexity of the language. Ones eliminated we use GF as framework for processing the language. This is made with creation of GF Abstract grammar that cover all needed sentences and GF concrete grammar for each language that we want to cover.

When the controlled language is used the main difficulty for the user is to write query that is part of the language. The auto complete is natural feature that inform the user what is the possibility for next words in the process of query creation. If there aren't any suggestions for auto complete the word or the phrase this will be a sign that the sentence is out of the controlled language.

# 2 THE ARCHITECTURE

The interoperability between NL and ontologies will provide human with natural interface for querying and retrieval of the semantic data. The main novelty of our approach is that user can make queries on all languages that are covered by the GF. The query will be transformed to GF Abstract Representation that is independent from the language then the abstract representation is converted to SPARQL query which is executed on the semantic repository. Then obtained results from the semantic search are transformed to the GF abstract representation from which the answer in natural language is retrieved.

---

<sup>1</sup> <http://gramaticalframework.org>

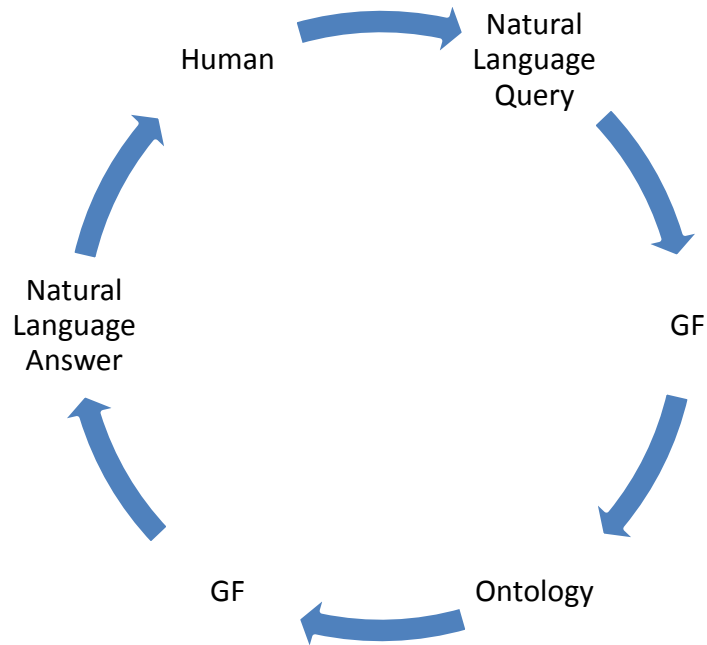


Figure 1. Question answering circle

The system is build with interaction between several modules:

- fsa module - module that is used for auto complete function. It build fsa from the possible strings that to be used for the auto complete function.
- lexicons - module that preprocess lexicons for the named entities at the queries.(Currently there are lexicons for people, organizations and locations.
- mapping rules module - module that is used for mapping between GF abstract representation and SPARQL
- natural-language-queries - module that use the modules described above to process the natural language query to SPARQL
- natural-language-answers - module that process the results from the SPARQL query and return results in the natural language.
- molto-web - the web UI for the project

### 3 AUTO COMPLETE TOOLS

The modules that realize the autocomplete function are the fsa module and lexicon module.

The fsa module build minimal acyclic final state automata from the given sentences to preprocess them and to be able to provide real time autocomplete for big set of sentences. For more implementation details about this please see the document "MOLTO - description of java projects, algorithms and feature"<sup>2</sup>.

<sup>2</sup> [svn://molto-project.eu/wp4/docs/molto-core-tools-description.pdf](http://svn://molto-project.eu/wp4/docs/molto-core-tools-description.pdf)

The lexicon module manage lexicons for the different named entities. The current lexicons that are used are for people, locations and organizations.

## 4 MAPPING RULES

The mapping-rules tool provide semi-automatic transformation from GF Abstract Representation to SPARQL. The transformation is realized with use of rules written at the language of the tool. You can find more details and examples about the syntax in APPENDIX A.

The mapping rules tool have dual use:

- preprocess - the rules are compiled and serialized at binary file that can be quickly load and use after that.
- real-time transformation - the tool is used from the natural-language-queries module for real-time transformation from GF Abstract representation of the query to SPARQL query.

## 5 RESULTS TO NATURAL LANGUAGE

The module for transformation of the GF results to natural language is named natural-language-answers. The module provide

- functionality for transforming a list of ontology files in nl format to a GF abstract and concrete grammar.
- real-time transformation of list of triples to natural language using the GF abstract and concrete grammar build on the previous step.

The automatic generation of GF grammar is tested on world knowledge base ontology that contains information about people, location 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 type of the instances. The predicates are used to provide the grammatical rules about the relations between objects. An example for automatic generated abstract grammar can be seen on figure 1. An example for automatic generated concrete English grammar can be seen on figure 3.

```
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 2. Automatic generated abstract grammar**

```

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 3. Automatic generated concrete English grammar

After we have generated the concrete English grammar it can be ported to other language with two steps - first change the grammar to use Resource Grammar Library for the new language and after that use lexicons to translate the words from English to the new language.

## 6 PROTOTYPE

This chapter describe the system prototype<sup>3</sup> that is built as part of D4.3. It is an information retrieval system that retrieve semantic data from the semantic repository. The data that is loaded in the repository is in the world knowledge base domain and include data for people, organizations, locations and job positions.

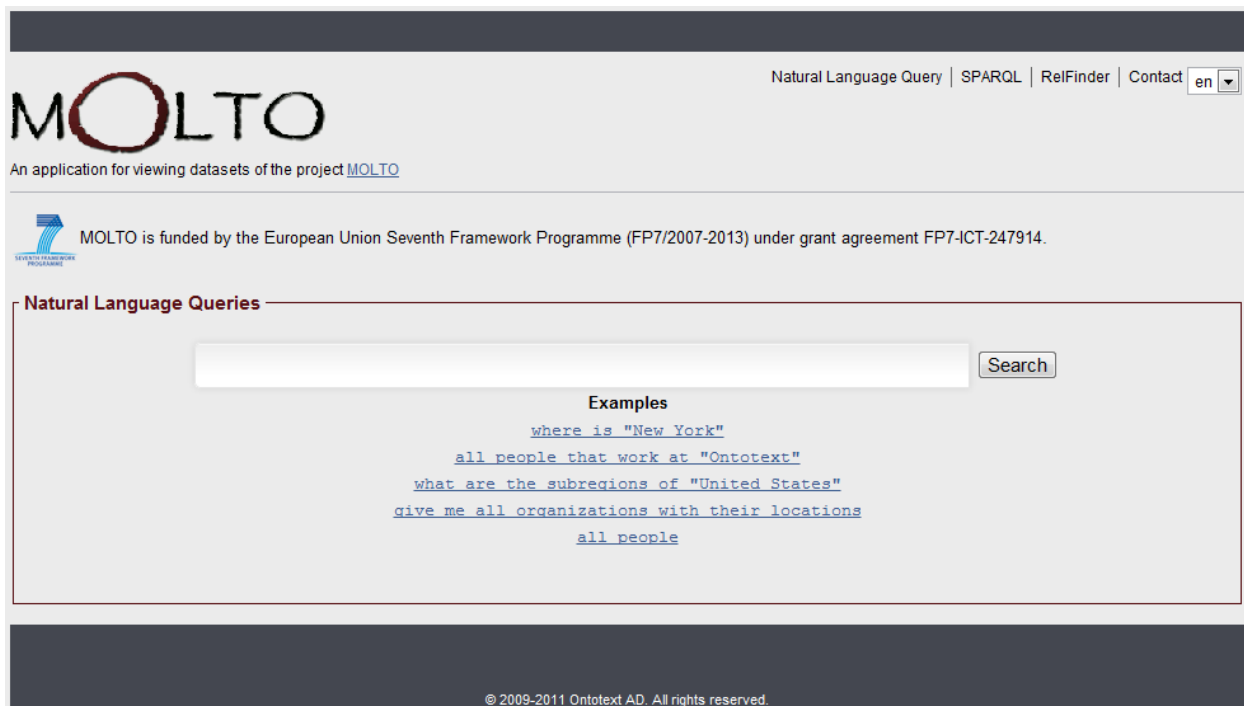


Figure 4. Main page of the prototype

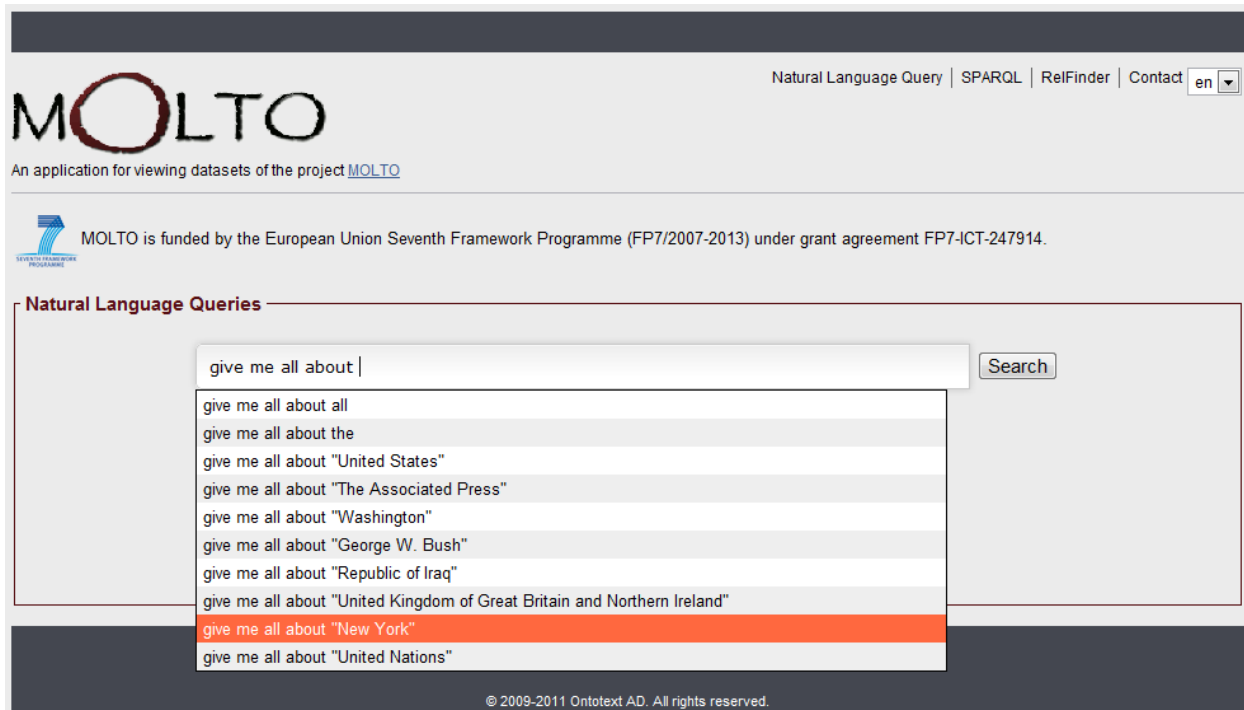
The main page offer the possibility of natural language search with the use of the controlled language of the system. You can see the user interface on figure 4. At the right up corner there is a

<sup>3</sup> <http://molto.ontotext.com>

drop down menu that can be used for change of the language of the interface and the query. There are 6 language that are already imported at the prototype. They are:

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

The auto complete function is available on all of the above written languages. For example of the auto complete see Figure 5.



**Figure 5. The auto complete example.**

After the user create his query he can execute it against the OWLIM<sup>4</sup> semantic repository. The returned results can be seen on the figure 6. The results are presented as several sentences on natural language and table with semantic results return from the semantic repository. Currently only English is covered as language for the returned results.

At the page there are the query in the natural language and link to the SPARQL representation of the query. The current representation of the natural language query "Give me all about New York" is:

```
construct WHERE {
```

---

<sup>4</sup> <http://www.ontotext.com/owlim>

```
?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.
}
```



Figure 6. Example for results from search

## 7 EXTENDING THE SCOPE

The prototype can be extended with:

- adding more databases
- extending the queries
- extending the answers
- add 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 (for example the sesame openrdf tools). Once added the data will be available for search and if the scope of the data is the same as the already implemented queries it will be retrieved as result.

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

Currently the GF grammar that is used for answers is automatically build from the ontology. It's not perfect but it can be manually manipulate to give better explanations. Another situation in which we can need to manipulate the GF grammar for the results is when the new databases are added or if we need to cover new language.

If we want to add new language for natural language queries at the prototype. We need a concrete GF grammar for it. It can be built using some of the already existing grammars as example and small modifications. After that we need to preprocess the sentences of the new language with the fsa module and to add it to the web interface.

## 8 CONCLUSIONS

At this document we present the prototype of using interoperability between the grammars and ontologies. The main advantages of this is the possibility to ask the semantic repository using natural language queries and to retrieve results again on natural language. This improve the accessibility of the system from the users that are not experts at the semantic repository domain and give a better chance of spreading.

## 9 FUTURE WORK

The described prototype will be used as a base for the D7.1 and D8.1 prototypes that are from the patents and cultural heritage domains.

## 10 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.

## 11 APPENDIX A - MAPPING RULES SYNTAX

There are several construction in the language:

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

this construction define the use of nameOfDefine(). All occurrences of nameOfDefine below the define will be replaced with the text the SELECT. This use of define is as in the c/c++ leanguages.

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

this define that all occurances of nameOFDefine() will be changed with "SELECT DISTINCT". The symbol ## is used to concatenate strings (this symbol will be used also rules below with the same meaning).

```
#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 define a mapping table with name nameOfTable. This table can be used in the rules for easy and flexible writing of different constants as URIs, names and etc.

```
//comment
```

this is the construction for comment

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

This construction is present one real rule. The rules parts are 3. First part is used as regex that try to match a GF Abstract Representation. The end of the first part and the beginning of the second part is marked with symbol |. The second part is used as boolean condition for execution of the rule. At the example ?X is marked with the rest of the Abstract Representation after the QSet word. The function single is used to determine if the X is tree or a single term. The function type can return blank string or Person, Organization, Location, JobTitle.

The third part of the rule determine the SPARQL query that match. At the example sparqlVar is a table that is have to be define at eh begining of the file and map the name of the variable X to a sparql variable. rdflabel() is define at the begining of the file too. At this example is used and the string ## which concatenate the name of the sparql variable with "\_label". This is very useful for dynamic creation of the names.