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:	30.09.2010
Actual date of delivery:	30.09.2010
Type:	Prototype
Status & version:	Final
Author(s):	Milen Chechev
Task responsible:	ONTO (WP4)
Other contributors:	

ABSTRACT

THIS DOCUMENT DESCRIBES THE MOLTO D4.3 PROTOTYPE HOSTED ON HTTP://MOLTO.ONTOTEXT.COM. IT GIVES MORE DETAILS ABOUT THE DIFFERENT MODULES THAT THE PROTOTYPE CONTAINS AND PROVIDES INSTRUCTIONS ABOUT THE EXPLOITATION OF THE PROTOTYPE.

CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

In everyday life we often look for answers of different questions and the most natural human way of representing the question is by using our natural language. The main problems when a program tries to retrieve answers of this kind of questions are:

(a) difficulties with understanding the natural language question,

(b) difficulties with processing more than one natural language,

(c) 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 faces the above problems. The prototype can process natural language questions in different languages, search for data in a semantic repository and present the retrieved data in the requested language. All this is possible because of the use of GF framework[1] and the interoperability between the GF grammar and the ontologies.

The prototype can work with natural language queries but also has some restrictions. The main restriction that is made is the use of 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. Once the ambiguity is eliminated we use GF as framework for processing the language. This is made with the creation of GF abstract grammar that covers all needed sentences and GF concrete grammars 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 a natural feature that informs the user what are the possible 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 the human with natural interface for querying and retrieval of the semantic data. The main novelty of our approach is that the user can make queries on all languages that are covered by the GF. The query is 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. Consequently, the obtained results from the semantic search are transformed to the GF abstract representation from which the answer in natural language is produced.

[1] http://gramaticalframework.org

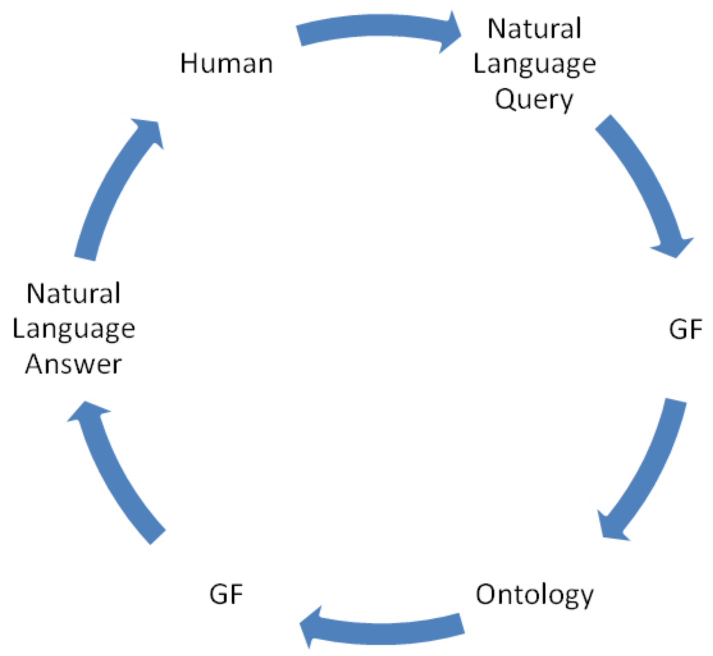


Figure 1. Question answering circle

The system is build with interaction between several modules:

- fsa module - module that is used for auto complete function. It builds fsa from the possible strings that to be used for the auto complete function.
- lexicons - module that preprocesses 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 uses 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 builds minimal acyclic final state automata from the given sentences to preprocess them and to be able to provide real time autocomplete for a big set of sentences. For more implementation details about this please see the document "MOLTO - description of java projects, algorithms and feature"[2].

[2] svn://molto-project.eu/wp4/docs/molto-core-tools-description.pdf

The lexicon module manages 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 in 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 as binary file that can be quickly loaded and used 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

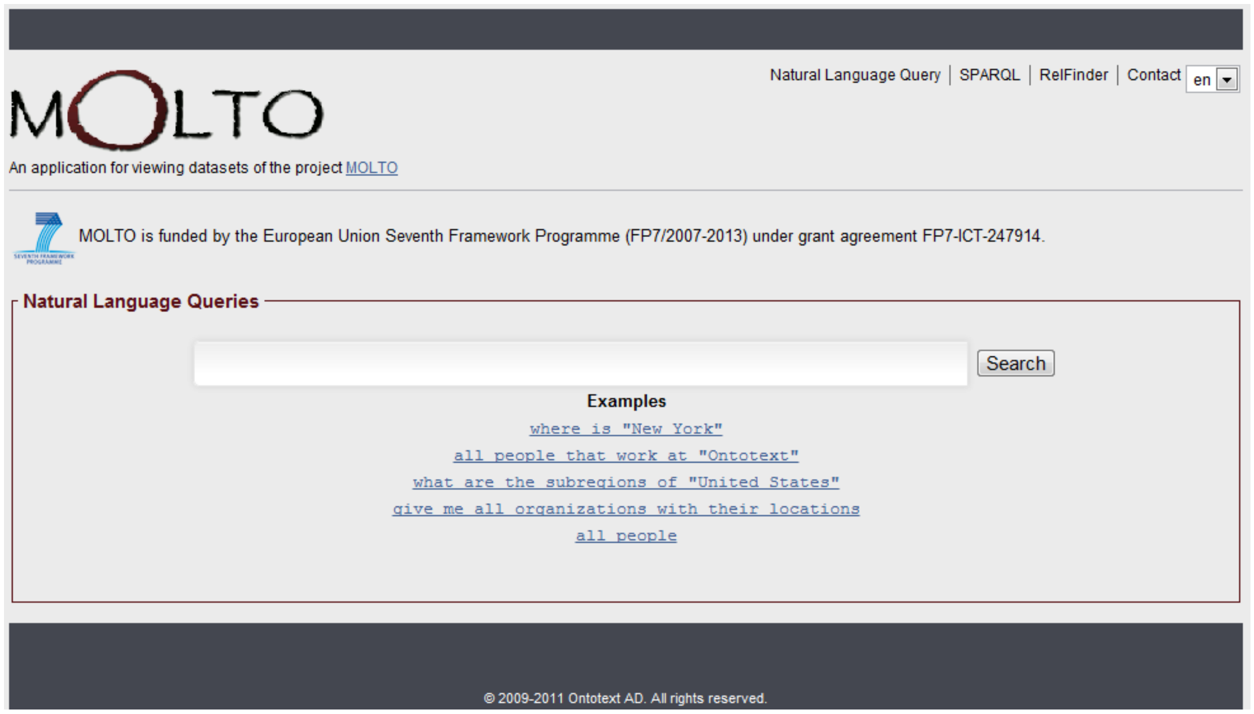
The module for transformation of the GF results to natural language is named natural-language-answers. The module provides:

- 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 the concrete grammar built in the previous step.

6 PROTOTYPE

This section describe the system prototype[3] that is built as part of D4.3. It is an information retrieval system that retrieves semantic data from the semantic repository. The data that is loaded in the repository is in the world knowledge base domain and include information about people, organizations, locations and job positions.

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



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

Examples

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

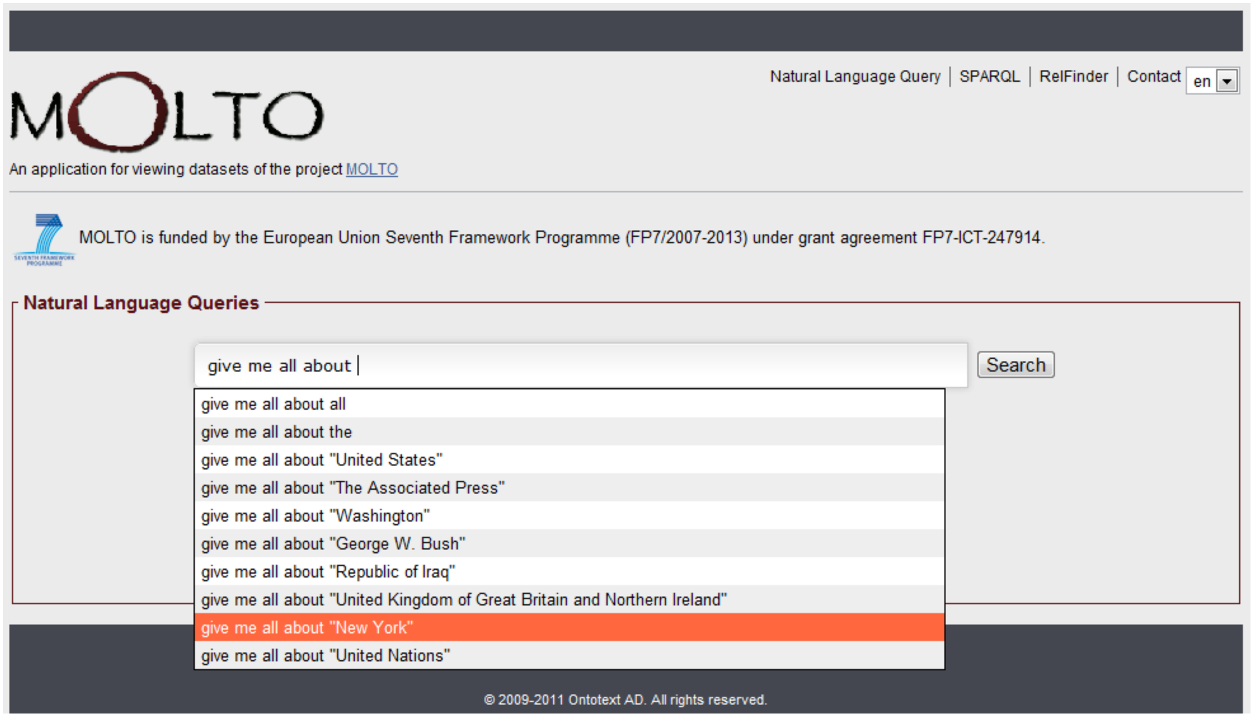
© 2009-2011 Ontotext AD. All rights reserved.

Figure 2. Main page of the prototype

The main page offers the possibility of natural language search with the use of the controlled language of the system. You can see the user interface on figure 2. At the right upper corner there is a 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 in the prototype. They are:

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

The autocomplete function is available on all of the above written languages. An example of the auto complete can be seen on Figure 3.



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.

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.

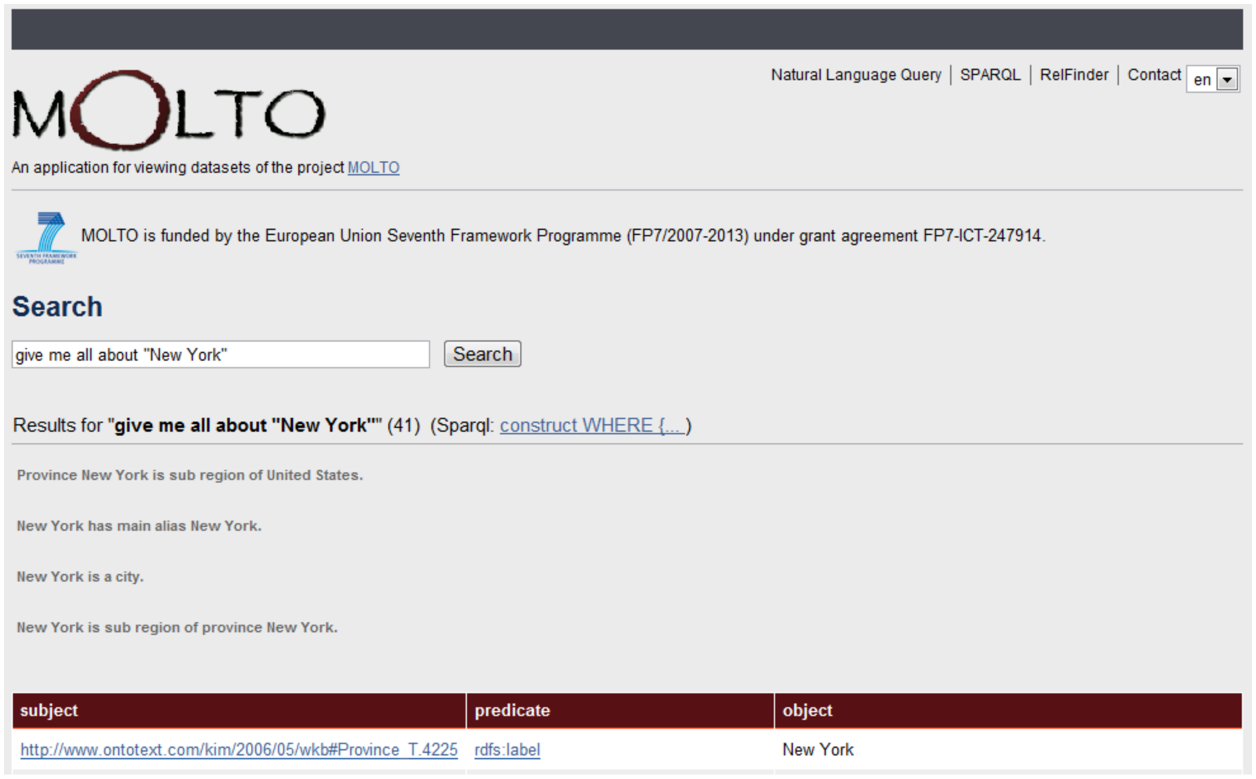
Figure 3. The auto complete example.

After the user creates his query he can execute it against the OWLIM[4] semantic repository. The returned results can be seen on the figure 4. The results are presented as several sentences in natural language and a table with semantic results returned from the semantic repository. Currently only English is covered as a language for the returned results.

At the page there are the query in natural language and a 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 {
?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 4. 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 sessame 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 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 built from the ontology. It's not perfect but it can be manually manipulated to give better explanations. Another situation in which we may need to manipulate the GF grammar for the results representation is when the new databases are added or if we need to cover a new language.

If we want to add a new language for natural language queries in the prototype, we need a concrete GF grammar for it. It can be built using some of the already existing grammars as example and applying small modifications to them. 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

We presented in this document the prototype of using interoperability between the natural language grammars and ontologies. The main advantages of this is the possibility to ask the semantic repository using natural language queries and to present the retrieved results again in natural language. This considerably improves the accessibility of the system from the users that are not experts in semantic web technologies and give a better chance of spreading of the usage of the semantic repositories.

9 FUTURE WORK

The described prototype will be used as a basis 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 constructions in the mapping language:

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

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

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

This defines that all occurances of nameOFDefine() will be changed with "SELECT DISTINCT". The symbol ## is used to concatenate strings (this symbol will be used also in the 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 defines a mapping table with the 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)) rdfslabel() sparqlVar(name(X)) ## "_label".
sparqlVar(name(X)) ?p ?o};
```

The above construction is the mapping rule. It contains 3 main parts. First part is used as regex that tries to match a 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 boolean condition for execution of the rule. In the example above ?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 a tree or a single term. The function type can return blank string or Person, Organization, Location, JobTitle.

The third part of the rule determines the SPARQL query that matches the selected GF Abstract Representation. In the example sparqlVar is a table that has to be defined in the begining of the file and to map the name of the variable X to a sparql variable. rdfslabel() is defined in the begining of the file too. In this example the string ## is also used. It concatenates the name of the sparql variable with "_label". This is very useful for dynamic creation of the variable names .