MOLTO

Multilingual Online Translation

Non multa, sed multum

Grammar-Ontology Interoperbility - Final Work and Overview

Contract No.:	FP7-ICT-247914
Project full title:	MOLTO - Multilingual Online Translation
Deliverable:	D4.3A Grammar-ontology interoperability - Final Work and Overview
Security (distribution level):	Public
Contractual date of delivery:	-
Actual date of delivery:	May, 2013
Type:	Annex to deliverable
Status & version:	Draft
Author(s):	Maria Mateva[1], Inari Listenmaa[3], Laura Toloşi[1], Ramona Enache[2], Aarne Ranta[2]
Task responsible:	Ontotext[1]
Other contributors:	UGOT[2], UHEL[3]

ABSTRACT
This document is D4.3A, an annex to the D4.3 deliverable of WP4 in the scope of the MOLTO project. It presents a final overview of the prototypes built in the scope of MOLTO, with respect to grammar-ontology interoperabilty. Also, it describes the further work on the topic, after M24 of the project, and gives some requested details on previous work. Next, the annex aims to address the reviewers remarks and recommendations from their last report. Finally, it serves to present a general overview of the achievements on the topic in MOLTO.

Contents

1 Introduction

Within MOLTO we have explored the interoperability between Gramatical Framework(GF)[1] and ontologies. Following [Con11] we have built three prototypes and surrounding tools which are described in [MI10], [Dam11], [CEDR12], and in the present document.

Semantic data is usually presented in the RDF standard[2], in which facts are described as sets of triples. Traditionally, sematic data is queried via the SPARQL[3] query language for semantic databases. Hence, the gap between the natural language of a common user and the semantic data remains open. Controlled languages come logically as a posible solution, since they provide strict interpretation of a natural language query. In the current document we present our exploration of the Gramatical Framework(GF)[4] as means to provide query language and response verbalization of the semantic repository data.

Given a specific ontology, our goal is to build a retrieval system that can be inquired in natural language with relevant quesions and also return answers in natural language. GF utilizes the process by providing abstract representation of human's query sentences that can be processed further. Additionally, it provides multilinguality at a comparatively low cost. Figure 1 demonstrates the workflow of the projected retrieval system.

In order to gain full interoperability between the grammars and the semantic datastore, we explored the following directions:

1. Given a GF abstract representation of a query sentence, generate corresponding SPARQL query

2. Given an ontology(RDF graph), create grammars for answers that will be generated from the RDF results

3. Given an RDF result, generate NL answer via the GF answer grammars

Mainly within WP4, but also WP7 and WP8, we have experimented with different approaches to utilize and possibly automate these steps. We have reached semi-automated SPARQL generation and template-based grammar generation. For the purpose, we have built tools and prototypes, which we present later in this document.

GF utilizes the NL query parsing to abstract representation, that is easier to convert to other languages and also fascilitates multilinguality. Most of the transitions of these steps are done automatically, see Figure 1. The transition between NL and GF concrete grammar is done automatically by the GF parser. The parser will also translate from GF concrete to GF abstract representations and from a call to GF concrete grammar to create a natural language answer. The semantic datastore, which in our case is OWLIM[5],

[1]http://www.grammaticalframework.org/
[2]http://www.w3.org/RDF/
[3]http://www.w3.org/TR/rdf-sparql-query/
[4]http://www.grammaticalframework.org/
[5]http://www.ontotext.com/owlim

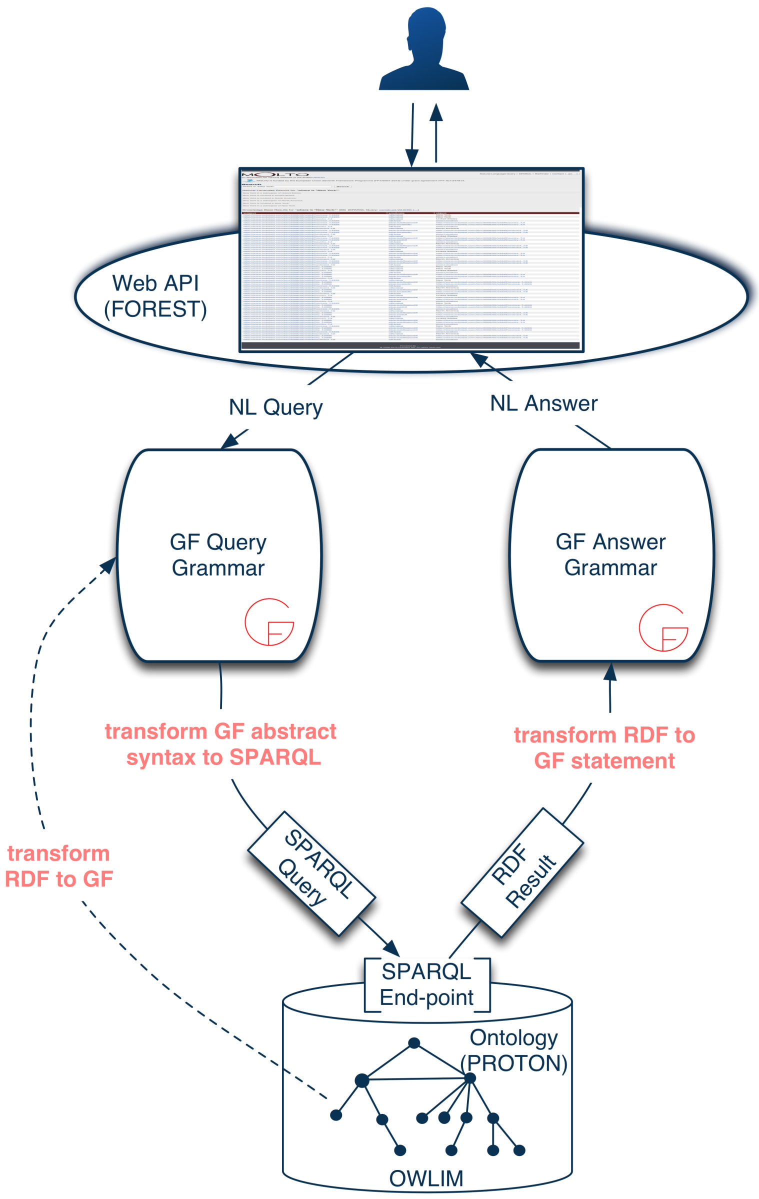


Figure 1: GF-Ontology interoperability

is responsible for the RDF answer to the SPARQL query.

The first and third from the above are very similar, as they seek to map an existing RDF graph to a GF grammar. Our aproach for this task was to create grammars that map entities from the ontology with the help of predefined templates. Details can be found in Section 3. The transition from the GF abstract representation to SPARQL query was explored in three different ways that are presented in Section 2. These include mapping rules between GF sytax trees and SPARQL queries, discursive patterns and exploring SPARQL as yet another ‘natural’ language.

Next, we have built a retrieval system prototype, called the (MOLTO) Knowledge Representation Infrastructure Prototype [6] that uses GF for natural language questions and answers support. It was the basis for two other MOLTO prototypes, that were explored and furtherly developed in the scopes of WP7 - the Patents Prototype[7] and WP8 - the Cultural Heritage Prototype[8].

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

[7] http://molto-patents.ontotext.com/

[8] http://museum.ontotext.com/

2 From GF Grammars to SPARQL

The first direction of interoperability to explore is how to obtain SPARQL queries, given specific GF grammars that define a query language. This task has no straightforward solution. We have explored two different approaches – mapping rules that make use of a predefined domain specific language and generation of SPARQL via GF translation, taking SPARQL as yet another query language.

2.1 Mapping Rules and FSA Autocomplete

Our initial approach for NL-to-SPARQL generation was to build a model of explicit mappings between GF abstract trees and SPARQL queries. This was achieved with mapping rules for which we built a domain specific language and a parser. No direct communication with GF service or process was used. Instead, we applied automatically generated trees and hand-written rules. As an addition, we built a custom autocomplete mechanism - finite state automata, which loads all previously generated trees as strings. It was built so that it uses lexicons for different types(e.g. *Person*, *Location*, *Organization*, *Drug*, etc)

Consequently, we found the mapping rules a redundant level of abstraction. Additionally, they did not explore GF in the best possible manner. The autocomplete based on FSA proved to be useful and fast enough, so it remained part of our MOLTO prototypes(See Section5). Next we give details on the implementations of this model.

2.1.1 Data lexicons

Data lexicons are typed lexicons, that are previously extracted from the semantic repository. These lexicons can be used both for the autocomplete module and in the concrete GF grammars. A good example of this scenario is the patents prototype([MGE+13]), where we have 8 dictionaries of Drugs, Active Ingredients, Patent Numbers, Application Numbers, Routes Of Administration, Dosage Forms, TE Codes, Markets. The list of entities is extracted from the aligned ontologies in the biomedical domain. They are integrated both in the grammars and in the autocomplete model. In the end we translated them in French and German(the original being English) with the SMT. This way we achieved completeness of the French and German query languages. Details are described in [MGE+13].

2.1.2 The old approach. Mapping rules

For the mapping rules, we built a small domain specific language and a parser for it. The language aims to provide syntactic sugar for some generalized mappings between GF and SPARQL. It is briefly described in [CEDR12]. An example to demostrate it, from the KRI prototype is the following rule

```
//all people with their aliases
//all organizations with their aliases
(QSet ?X) | single(X) && type(X) == "" && name(X) != Location —>
```

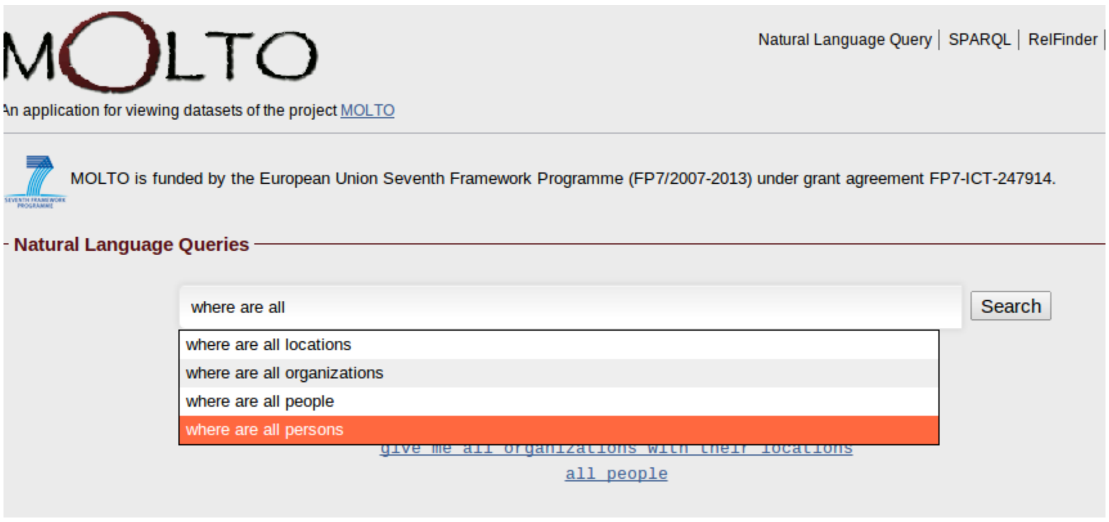
```
construct WHERE {
  sparqlVar(name(X)) rdftype() class(name(X)) .
  sparqlVar(name(X)) property(hasAlias) sparqlVar(name(X)) ## "_alias". };
```

In this example, the left-hand side is the GF parsed tree syntax *(QSet ?X)*. *name(X)* gives the type of the entity X, e.g. *Location*, *Organization*, *Person*, *Job Title*. On the right-hand side of the rule we declare a "construct" that maps the *QSet* sententces. In the above example, we use *sparqlVar* to get the name of the variable(e.g. *?name*), the *class* function to return its RDF type, and the *property* function to return full name of the "hasAlias" predicate.

The example demonstrates how we make dynamic generation of the SPARQL query, depending on its type, which is inferred via the type of lexicon we allow for this abstract GF tree.

2.1.3 Autocomplete with GF and FSA

The three prototypes of Ontotext use custom autocomplete, which is based on a minimalistic acyclic finite state automaton and predefined GF queries. They use preliminary generated resources from the GF grammar of target. These are mostly comprized by the generated sentences from a grammar and data lexicons for the separate data entity types. In particular - these are the data types that will be retrieved from the RDF database. The lexicons are useful for the generation both of this model and of the GF grammar itself. In fact, the initial motivation for using FSA instead of the GF autocomplete, was the possibility of a very large semantic repository, whose entities would be too numerous for GF to support. Consequently, we found FSA very useful because of the control it gives to the displayed queries and also the control of suggested types.


MOLTO
Natural Language Query | SPARQL | RelFinder |
An application for viewing datasets of the project MOLTO
Natural Language Queries
where are all
Search
where are all locations
where are all organizations
where are all people
where are all persons
give me all organizations with their locations
all people

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

Figure 2: MOLTO KRI autocomplete in action

2.1.4 Advantages and disadvantages

The mapping rules allow dynamic generation of SPARQL to certain extent, but they are difficult to debug and maintain, when changes in the grammar or in the desired SPARQL syntax. The parser for the rules also required support. Moreover, this model does not benefit from GF as much as it is possible. For example, GF can be used for direct translation to SPARQL and, hence, the need of the mapping rules and their parser invokation become redundant.

2.2 GF as means to generate SPARQL. YAQL

Through the times of MOLTO an iteresting idea was explored by consortium partners - to review SPARQL as a natural language from GF point of view. This means that in addition to abstract and concrete grammars for all natural languages, we create a SPARQL concrete grammar as well. In a way, we observe SPARQL as a natural language to which we need to translate.

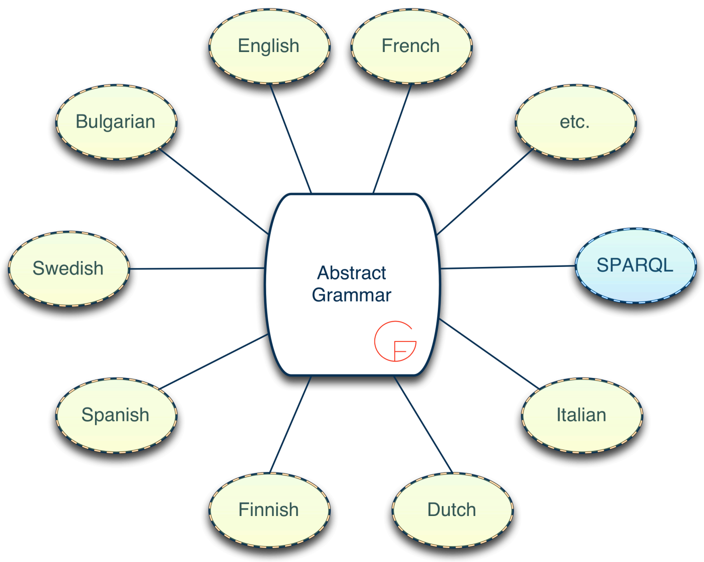


Figure 3: SPARQL as another GF language

2.2.1 YAQL

In the final year of MOLTO, UGOT contributed additionally to WP4 with the creation of the YAQL("yet another query language") grammar module that provided basis for domain specific SPARQL generation. The implementation of a query language implementation in YAQL is explained in [Ran12]. YAQL has straightforward abstract syntax generation from ontology, with just the minimum of lexical types:

```
        Kind : usually CN
        Entity : usually NP
        Property : can be VP, AP, ClSlash
        Relation : VPSlash built from V2, AP, comparatives
```

2.2.2 SPARQL generation model - WP7

The SPARQL generation model of WP7 was described in [MGE+13]. It is the first implementation of SPARQL generation via GF that we applied in a KRI-based prototype. The model is simple - it just allows mapping a GF function(abstract query expression) to a SPARQL query. The below example shows the astract, NL concrete and SPARQL concrete representations of the “show patents that mention X and Y” query as developed for the PatentQuery[9] grammar.

Abstract syntax:

```
    QShowConceptXY : Concept -> Concept -> Query ;
```

Concrete natural language interface:

```
    QShowConceptXY c1 c2 = mentionP (mkNP and_Conj c1 c2) ;
```

Concrete SPARQL syntax:

```
    QShowConceptXY c1 c2 =
    {s = "PREFIX pkm: <http://proton.semanticweb.org/protonkm#> $n
        PREFIX psys: <http://proton.semanticweb.org/protonsys#> $n
        CONSTRUCT { $n ?doc pkm:mentions ?x . $n  ?doc pkm:mentions ?y } $n
        WHERE { $n  ?x psys:mainLabel " ++ c1.s ++". $n
        ?doc pkm:mentions ?x . $n ?y psys:mainLabel " ++ c2.s ++". $n
        ?doc pkm:mentions ?y . $n  }  " } ;
```

The SPARQL generation in this case is similar to the mapping rules, but it relies on GF for translation and does not deal with additional layer of predefined rules.

2.2.3 SPARQL generation model - WP8

The SPARQL generation model of WP8 was described in [DRE+13] and [DDE+13]. It is more dynamic and allows parametrized building of the SPARQL queries. The below example is from the *QueryPainting* grammar[10], *QueryPaintingSPARQL.gf*:

```
  QPainter p = {wh1 = "?painter"; prop = p ; wh2 = " painting:createdBy ?painter ."} ;
  QYear p = {wh1 = "?date"; prop = p ; wh2 = " painting:hasCreationDate ?date ."} ;
  QMuseum p = {wh1 = "?museum"; prop = p ; wh2 = " painting:hasCurrentLocation ?museum ."} ;
  QColour p = {wh1 = "?color"; prop = p ; wh2 = " painting:hasColor ?color ."} ;
```

[9]svn://molto-project.eu/wp7/query/grammars
[10]svn://molto-project.eu/wp8/d8.3/grammars

```
QSize p = {wh1 = "?dim"; prop = p ; wh2 = " painting:hasDimension ?dim ."} ;
QMaterial p = {wh1 = "?material"; prop = p ; wh2 = " painting:hasMaterial ?material ." } ;
```

We believe this is the direction in which SPARQL generation from GF grammars should develop in a certain use case. Similar ideas were exploited also in WP6, with Sage syntax generation in WP11, ACE syntax generation, and in WP12, in the Be Informed Studio with the business modeling domain grammars.

2.3 Creation of a query language

Our experience in the KRI-based prototypes shows that the better the allowed questions cover actual realtions in the ontology graph, the better the query language is. This gives us motivation to search for query patterns that cover well the predicates in the ontology. The verbalization methodology suggested in 3.2 can utilize automatic example-based creation of queries for the ontology by selecting probable GF patterns. Due to the variety of possible question types, this approach can be semi-automatic in our best expectations.

2.4 Discussion on Levels of Automation

Adopting GF as described in Section 2.2 allows direct translation from natural language to SPARQL(or any other machine language, such as ACE or Sage languages). The level of automation depends on the complexity of the required query language and on the implementation of the SPARQL generation grammars. Hence, a more dynamic approach will require more efforts of GF experts. Undisputedly, this model is preferable to the earlier approach of mapping rules.

3 GF Grammars Generation from Ontology

In the scope of WP4, we have chosen to represent ontologies by RDF. The reason for this decision is that each fact in the RDF graph has its own semantics that is easily conceivable. It is defined by a triple of `subject–predicate–object`, which we aim to verbalize as `SUBJECT is PREDICATE of OBJECT.`, or `SUBJECT HAS-PREDICATE OBJECT.`, or `SUBJECT VERB-PREDICATE OBJECT.`

GF is very suitable for verbalizing sentences of this kind, and also for providing multilingual representation of the same .

3.1 The GQHB Tool and the Grammar Ontology Helper for the GF Eclipse Plugin

The GQHB tool, described in [CEDR12] was extended to wizard as part of the GF Eclipse plugin. This work belongs to WP2 but was not mentioned in its deliverables, due to its late completion. Manual for the usage of the wizard is publicly available at `https://github.com/GrammaticalFramework/gf-eclipse-plugin/blob/master/README_ontology-grammar.md`.

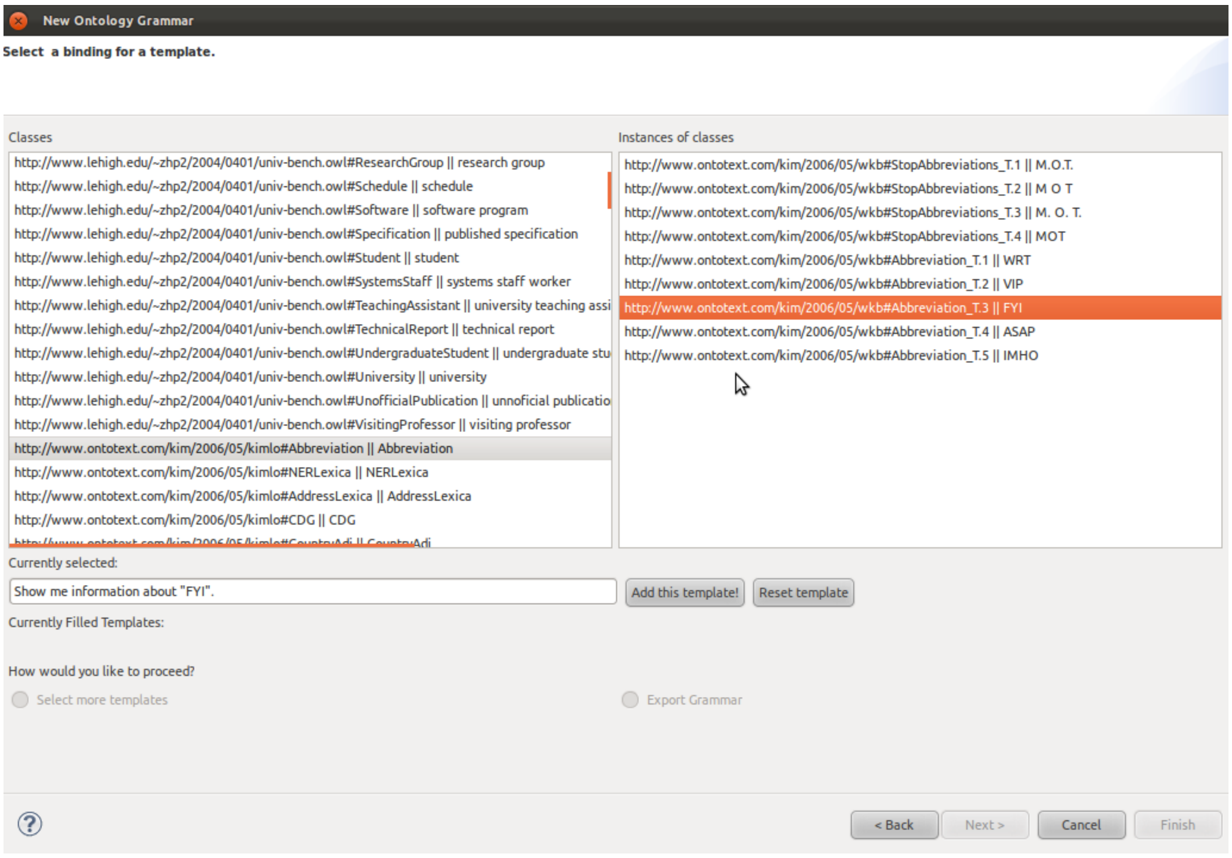
The wizard has almost identical functionality as the GHQB tool. It connects to a SPARQL endpoint, inspects the ontology and provides listst of classes and the respective instances of these classes. Then, it creates GF categories from the selected classes, GF entities of these categories, that are verbalized with the labels of the instances of the classes. This generation is based on a template that should have previously been defined by GF expert. The structure of the xml template is:

```
– "templates" – root element
    – "template" – template for a single query
    – "pattern" – the query displayed to the user
    – "fun" – abstract grammar`s initial functions(without the generated ones)
    – "lincat" – concrete (English) grammar`s initial linearization categories
    – "lin" – concrete (English) grammar`s initial linearizations
    – "sparql" – SPARQL query, usually with pattern
    – "sparql–lin" – concrete SPARQL grammar`s initial linearizations
```

A screenshot of the wizard's GUI is shown on Figure 4.

An example of the template can be found at `svn://molto-project.eu/wp4/projects/molto-core/molto-repository-helper/resources/template.xml`. In the end three concrete grammars are generated: English concrete(can be any other language), SPARQL concrete and an abstract grammar.

The motivation for such a tool and wizard is to enable GF non-experts to create GF grammars on-line. Anyway, the creation of the templates requires significant GF expertise for creation of generic grammar model.



Select a binding for a template.

Classes

```
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#ResearchGroup || research group
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Schedule || schedule
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Software || software program
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Specification || published specification
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student || student
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#SystemsStaff || systems staff worker
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#TeachingAssistant || university teaching assi
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#TechnicalReport || technical report
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#UndergraduateStudent || undergraduate stu
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#University || university
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#UnofficialPublication || unnoficial publicatio
http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#VisitingProfessor || visiting professor
http://www.ontotext.com/kim/2006/05/kimlo#Abbreviation || Abbreviation
http://www.ontotext.com/kim/2006/05/kimlo#NERLexica || NERLexica
http://www.ontotext.com/kim/2006/05/kimlo#AddressLexica || AddressLexica
http://www.ontotext.com/kim/2006/05/kimlo#CDG || CDG
http://www.ontotext.com/kim/2006/05/kimlo#CountryAdj || CountryAdj
```

Instances of classes

```
http://www.ontotext.com/kim/2006/05/wkb#StopAbbreviations_T.1 || M.O.T.
http://www.ontotext.com/kim/2006/05/wkb#StopAbbreviations_T.2 || M O T
http://www.ontotext.com/kim/2006/05/wkb#StopAbbreviations_T.3 || M. O. T.
http://www.ontotext.com/kim/2006/05/wkb#StopAbbreviations_T.4 || MOT
http://www.ontotext.com/kim/2006/05/wkb#Abbreviation_T.1 || WRT
http://www.ontotext.com/kim/2006/05/wkb#Abbreviation_T.2 || VIP
http://www.ontotext.com/kim/2006/05/wkb#Abbreviation_T.3 || FYI
http://www.ontotext.com/kim/2006/05/wkb#Abbreviation_T.4 || ASAP
http://www.ontotext.com/kim/2006/05/wkb#Abbreviation_T.5 || IMHO
```

Currently selected: Show me information about "FYI".

Currently Filled Templates:

How would you like to proceed?

Figure 4: Grammar-Ontology Wizard - Selection of instances of classes to generate GF grammars

3.2 Generation of GF grammars from RDF triples

Throughout the work on MOLTO KRI prototype(see Section 5.1), and inspired by the idea of a generic grammar, described in [CEDR12], we created a generator of generic grammar that explores ontology facts and generates grammars for them. The automatic generation of grammar entities and semi-automatic generation of grammars for the RDF predicates are

The idea continues from earlier work within the MOLTO project. [Ena10] describes the translation of the SUMO[11] axioms to GF, the linearizations written mostly manually. Our approach automatizes some of the work, taking advantage of the usually consistent naming conventions of the RDF predicates. [Lis12] is a feasibility study which describes different uses of ontologies in lexicon management. As a conclusion, generation of GF grammars from RDF data is found promising. Other directions taken by [Lis12] and the University of Helsinki include pure lexicon generation; see section 3.4.

We use the RDF's objects and subjects with their truncated URIs and labels, to create

[11]Suggested Upper Merged Ontology

respectively abstract and English concrete GF representations. The verbalization of predicates was further investigated in the final year of MOLTO. It has also been of scientific interest to external research groups([AGL13]).

The motivation for this was that RDF predicates are usually meaningful and follow strict naming convention, especially in a single ontology. This is not always true and sometimes RDF predicates names are in different languages(e.g. in DBpedia[12]).

Hereby we present a method to verbalize RDF predicates, in case that they are in English and follow a naming convention. It is based on in-depth analysis of 1700+ predicate names from 10 random public repositories. It is worth pointing that the repositories were not preselected in any manner. A number of observations and heuristic led to the separation of the predicates into groups that allow easier verbalization via example-based GF grammars.

For predicates in other languages we could utilize similar algorithm.

3.2.1 Subjects and Objects Representation

For RDF `subjects` and `objects` GF representation can be made straight-forward - all are presented in GF by the `Entity` category. This model can further extended, for example by defining more specific type of the Entity and hence by have lots of categories(*Person*, *Location*, etc.). We tried this deferentiation in the MOLTO KRI prototype(See Section 5.1) but we found no practical benefit in splitting the categies into the grammar, except for easier lexicon splitting, which utilizes for the autocomplete technology.

The corresponding anstract and concrete grammar representation for the above example is as follows for the first five PROTON class instances(alphabetical order) is:

Abstract syntax:

```
cat Entity;

fun
    Airline_T_1 : Entity;
    Airline_T_10 : Entity;
    Airline_T_10_0 : Entity;
    Airline_T_10_1 : Entity;
    Airline_T_10_2 : Entity;
    ...
```

Concrete syntax(data from the English labels):

```
lincat
    Entity = Str ;

lin
    Airline_T_1 = "Japan Airlines System Corporation" ;
    Airline_T_10 = "Cathay Pacific Airways, Ltd." ;
    Airline_T_10_0 = "Cathay Pacific Airways, Ltd." ;
    Airline_T_10_1 = "Cathay Pacific Airways, Limited" ;
    Airline_T_10_2 = "Cathay Pacific Airways" ;
```

[12] http://dbpedia.org/

```
    ...
```

The Entity category serves to represent ontologies subjects and predicates in GF. Anyway, to achive complete automation of the GF verbalization of an RDF triple, we need to focus on the predicates. In the case of MOLTO KRI we provided representation for a number of useful predicates. They were chosen by human after observation of the graph results to each `construct` query[13].

Normally the user would not need to verbalize all predicates. For example, `rdf:type` is one we prefered to skip, as "Borislav Popov has type Person." is useless information in our case.

3.2.2 Predicates Representation and Predicates Types

The abstract representation clearly has automatable parts - e.g. the one that lists the predicates.

```
cat
    Phrase;
    Property;
    [Property]{2};

fun
    text: Entity -> Property -> Phrase;
    and: [Property] -> Property;

    activeInSector: Entity -> Property ;
    childOrganizationOf : Entity -> Property ;
    hasCapital : Entity -> Property ;
    holder : Entity -> Property ;
    locatedIn : Entity -> Property ;
    owns : Entity -> Property ;
    ...
```

The corresponding concrete English syntax to the above also reveals repetable behaviour. Examples to defend the statement are the GF pattern for `activeInSector` and `locatedIn`, as well as the one for `hasCapital` and `holder`. The latter is verbalized as "has holder".

```
lincat
    Phrase = S;
    Property = VPS;
    [Property] = [VPS];

oper s2e : Str -> NP = \s -> symb (mkSymb s) ;

lin
    BaseProperty = BaseVPS;
    ConsProperty = ConsVPS;
    and ps = ConjVPS and_Conj ps;
    text x y = PredVPS (s2e x) y;
```

[13]`construct` queries return graph over certain description. Hence, they always return exactly RDF triples.

```
activeInSector x = MkVPS (mkTemp presentTense simultaneousAnt)
  positivePol (mkVP (mkA2 (mkA "active") (mkPrep "in")) (s2e x)) ;

hasCapital x = MkVPS (mkTemp presentTense simultaneousAnt)
  positivePol (mkVP (mkVPSlash have_V2) (mkNP (mkCN (mkCN (mkN "capital")) (s2e x)))) ;

holder x = MkVPS (mkTemp presentTense simultaneousAnt)
  positivePol (mkVP (mkVPSlash have_V2) (mkNP (mkCN  (mkCN (mkN "holder")) (s2e x)))) ;

locatedIn x = MkVPS (mkTemp presentTense simultaneousAnt)
  positivePol (mkVP (mkA2 (mkA "located") (mkPrep "in")) (s2e x)) ;

owns x= MkVPS (mkTemp presentTense simultaneousAnt)
  positivePol (mkVP (mkV2 "own") (s2e x)) ;
...
```

Having the answer grammar generated, what remains is to have it verified it by an expert. Afterwards, a call to it is made for each RDF triple from a result set that has to be verbalized. Our MOLTO KRI prototype uses the below verbalization:

```
l text <subject> (<predicate> <object>)
```

The final answers are passed to the Forest UI. Examples are shown on Figure 5.

In order to have reliable verbalization of RDF predicates, we had to find patterns that we could apply in the common cases for predicate names. Hence, division of predicates, according to a GF pattern compatible type is needed. We explored 10 public repositories with about 1700 unique predicates, made analysis of the common predicates patterns and extracted several most common patterns of predicate names. The initial research made it clear that predicates can be divided into three general groups and about 20 smaller ones.

For those that do not fit, we suggested a compromise pattern that has to work for the general pattern. Of course it would be great to have these refined by GF expert.

Below we list the three general types with their GF generalization, where all predicates found place, after stable initial preprocessing.

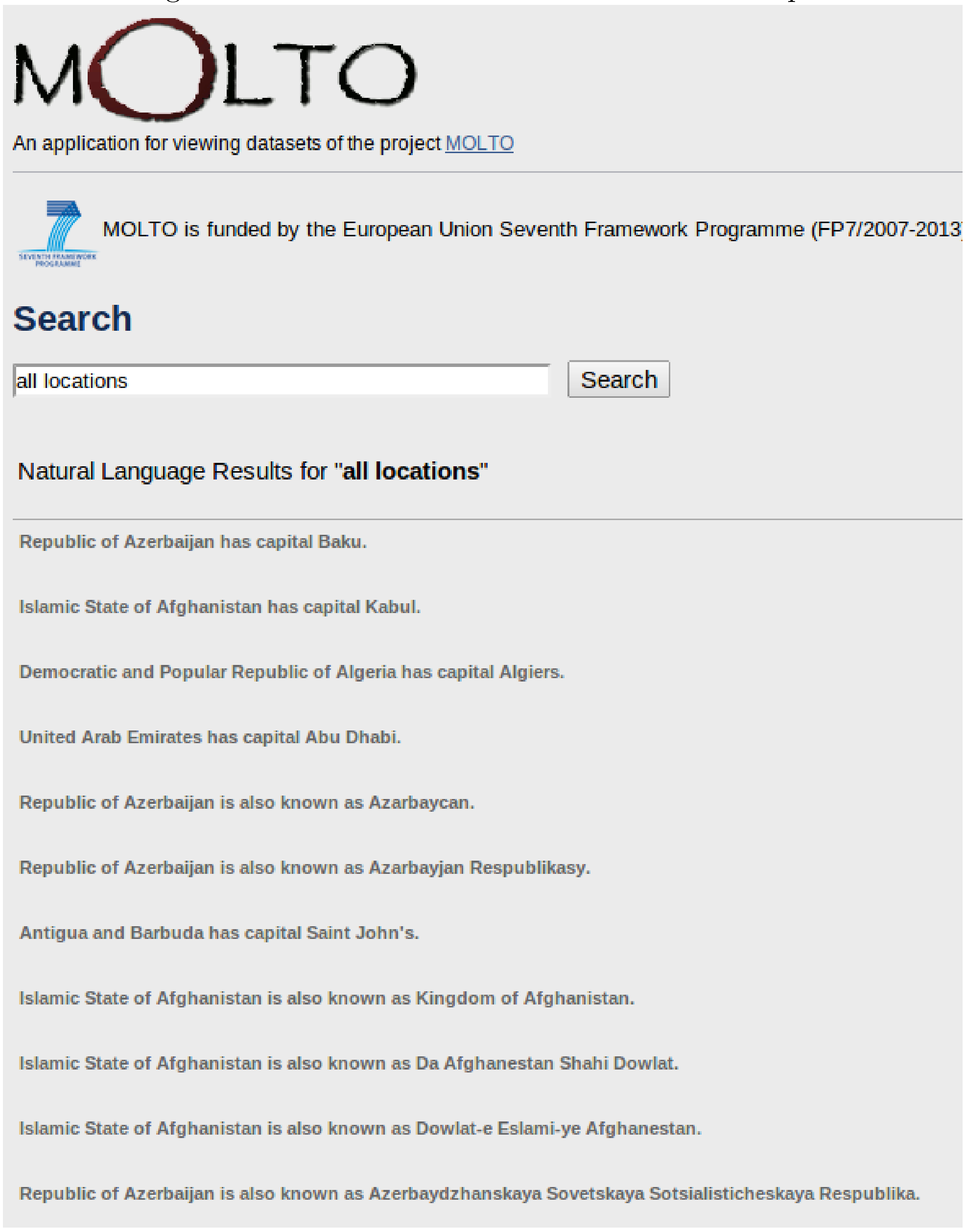
Group name	Summary	Description
A	"HAS" predicates	Predicates that are most easily described by "has `quality`" phrase.
B	"IS" predicates	Predicates that are most easily described by "is `quality preposition`" phrase.
C	Other verbs group	Predicates that start with verbs other than forms of "to be" and "to have".

Table 1: General groups of RDF predicates

Every predicate was normalized to one of the above classes. For example, all noun-prepositions, e.g. "parentOf" were normalized to "is parent of". All noun-nouns, e.g. "systemProperty" were normalized to "has system property".

Examples of GF patterns:

Figure 5: MOLTO KRI: verbalizaton of RDF triples


MOLTO
An application for viewing datasets of the project MOLTO
SEVENTH FRAMEWORK PROGRAMME
MOLTO is funded by the European Union Seventh Framework Programme (FP7/2007-2013
Search
all locations
Search

Natural Language Results for "**all locations**"

Republic of Azerbaijan has capital Baku.

Islamic State of Afghanistan has capital Kabul.

Democratic and Popular Republic of Algeria has capital Algiers.

United Arab Emirates has capital Abu Dhabi.

Republic of Azerbaijan is also known as Azarbaycan.

Republic of Azerbaijan is also known as Azarbayjan Respublikasy.

Antigua and Barbuda has capital Saint John's.

Islamic State of Afghanistan is also known as Kingdom of Afghanistan.

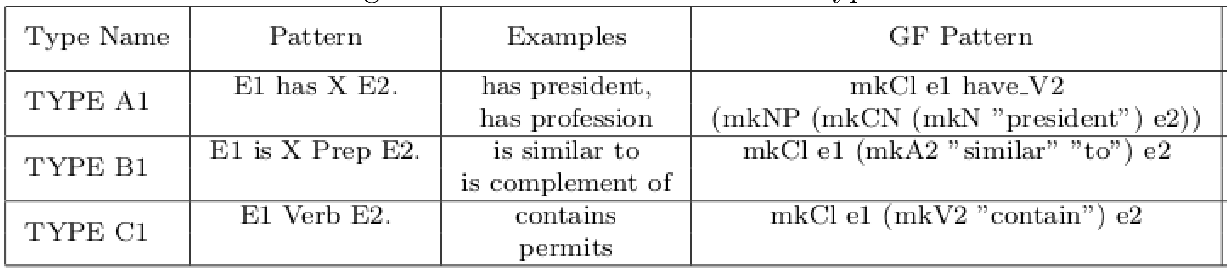
Islamic State of Afghanistan is also known as Da Afghanestan Shahi Dowlat.

Islamic State of Afghanistan is also known as Dowlat-e Eslami-ye Afghanestan.

Republic of Azerbaijan is also known as Azerbaydzhanskaya Sovetskaya Sotsialisticheskaya Respublika.

In the patterns described below, the N keyword stands for both nouns and adjectives. The motivation for this assumption is that in English the noun-noun phrases and adjective-noun phrases behave similarly. For example "system properties" and "useful properties" behave in the same way in the sentence. Unfortunately, this assumption is incorrect for languages like German, French, Finnish, Bulgarian and many others, where one could expect significant differences like reverse of order or need of gender alignment. Hence, if this direction of preprocessing is kept in the future, some grammatical analysis will be

Figure 6: Patterns to verbalize GF types



Type Name	Pattern	Examples	GF Pattern
TYPE A1	E1 has X E2.	has president, has profession	mkCl e1 have_V2 (mkNP (mkCN (mkN "president") e2))
TYPE B1	E1 is X Prep E2.	is similar to is complement of	mkCl e1 (mkA2 "similar" "to") e2
TYPE C1	E1 Verb E2.	contains permits	mkCl e1 (mkV2 "contain") e2

necessary for languages other than English. It would split the example types in even larger number.

Nevertheless, our observations are that the majority of predicates fit in few of the types described in Appendix A, Table 2 E1 and E2 stand for entities as described in the previous section.

The obvious problems with this approach are that we cannot guarantee complete correctness, as ontologies are designed by humans and can have many name convention floats, or just new predicate structures. Also, predicate semantics are sometimes ambiguous. For example, *A–contains–B* can be a design to both "A contains B." and "B contains A." as explained in ([MS06]).

3.2.3 Software Implementation

We have implemented a utility in java that takes a SPARQL end point as a parameter and generates four GF files that represent automatically generated English grammar. This architecture was derived from the Wkb grammar, created in collaboration by Ontotext and UGOT. It was mostly automatically generated and manually editted by GF expert. The source of this grammar is publicly available at `svn://molto-project.eu/wp4/projects/molto-kri/natural-language-queries/resources/grammars/answer` and consists of:

- Abstract GF grammar for the predicates
- Concrete English GF grammar for the predicates
- Abstract GF grammar for the data(entities from the ontologies - both subjects and objects)
- Concrete English GF grammar for the data - with the English labels

3.3 NL description generation from semantic results. Museum use case

Internally we have evaluated that verbalization of simple facts is not quite exciting it we cannot generate whole object description.

3.3.1 NL answer generation from semantic results

One option for object description is to verbalize all facts in the molecule of the concrete ontology instance. This means its surrounding in the ontology graph with certain depth. For example, on Figure 5 we have both

- Islamic State of Afghanistan has capital Kabul.
- Islamic State of Afghanistan is also known as Kingdom of Afghanistan.

This way we could verbalize all triples returned to

```
construct where {
   ?subject ?predicate ?object .
   ?subject rdfs:label "Islamic State of Afghanistan" .
}
```

which is the molecule of "Islamic State of Afghanistan" with depth 1.

3.3.2 Object Description. Museum Use Case

In the WP8 prototype on cultural heritage, we generate paintings descriptions([DRE+13]). For each painting, we execute a single call to GF description function which takes the following types of parameters

- painting(URI)
- painter
- title
- length
- height
- year
- material
- museum

The natural language queries are translated to SPARQL queries against a single ontology, *painting.owl*. Each query that generates description is a `select` query with (a subset of) the above list of parameters. The base query, that answers "show everything about all paintings" is

```
PREFIX painting: <http://spraakbanken.gu.se/rdf/owl/painting.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

#common parameters list for description generating queries
SELECT DISTINCT ?painting ?title ?material ?author ?year ?length ?height ?museum
WHERE   {
?painting rdf:type painting:Painting ;
    rdfs:label ?title ;
        painting:createdBy ?author;
        painting:hasCurrentLocation ?museum;
        painting:hasCreationDate ?date;
        painting:hasDimension ?dim  .
    ?author rdfs:label ?painter .
    ?date painting:toTimePeriodValue ?year .
    ?dim painting:lengthValue ?length ;
        painting:heightValue ?height .
    ?museum rdfs:label ?loc .
} LIMIT 200
```

The answer generating call to the GF `DPainting` is of the form:

```
-unlextext -lang="TextPaintingEng" DPainting (PTitle
  TAdoration_of_the_Magi__28Vel_C3_A1zquez_29)
 PDiego_Velazquez NoPaintingType NoColours (MkSize (SIntInt 127 204)) (MkMaterial MCanvas) (
     MkYear (YInt 1619)) (MkMuseum MMuseo_del_Prado)
```

The above GF request generates the following English description.

```
"The Adoration of the Magi was painted on canvas by Diego Velazquez in 1619. It measures 127 by
     204 cm. This work is displayed at the Museo del Prado."
```

Except for English, we support 14 other languages, whose grammars were created manually.

3.4 Lexicon generation with TermFactory

As a complementary strategy for grammar creation, UHEL has focused on lexicon generation. Lexicons translation can very well enforce the automatic grammar generation for different languages, when we have a base for English(or other language). UHEL has defined a format from which there is a conversion to GF lexicon. The format is used in TermFactory, a platform for multilingual terminology management created in UHEL.

TermFactory is a decentralized system which combines terminologies and ontologies from many sources. A TermFactory site is a web site identified by a site URL, owned by some organization, that maintains one or more TF back end repositories and one or more TF front end platforms/tools. A TF repository consists typically of one or more web servers with file space, one or more ontology databases, and TermFactory web services. Each site stores those ontologies which it owns and manages, in addition it can cache or mirror ontologies which are owned by some other site. The triples from different sources are linked to TF top ontology, whose structure is explained later.

TF frontends include SPARQL endpoint and an editor for modifying the entries .As for concrete scenarios, [Lis12, pp. 29–30] describes some of the functions of TermFactory:

"A possible scenario is that we have an ontology that has no name predicate, that is, the concepts are represented by URIs or other identifiers not directed to human readers. In TF we do not modify the original ontology–it need not even be physically located on the TF server. Instead, we make a local extension on TF platform, enhancing the ontology with the triples we need, in this case names or labels in natural language(s)."

The structure of the TermFactory top ontology schema is shown in figure 7. Baseforms (yellow) are linked to expressions (yellow), and expressions are linked to concepts (blue) with the relation in green. The green relation is handled with the triples whose subject is of type `term1:en-dog-N_-_ont-dog`; the predicate `term:hasDesignation` marks the expression and `term:hasReferent` the concept.

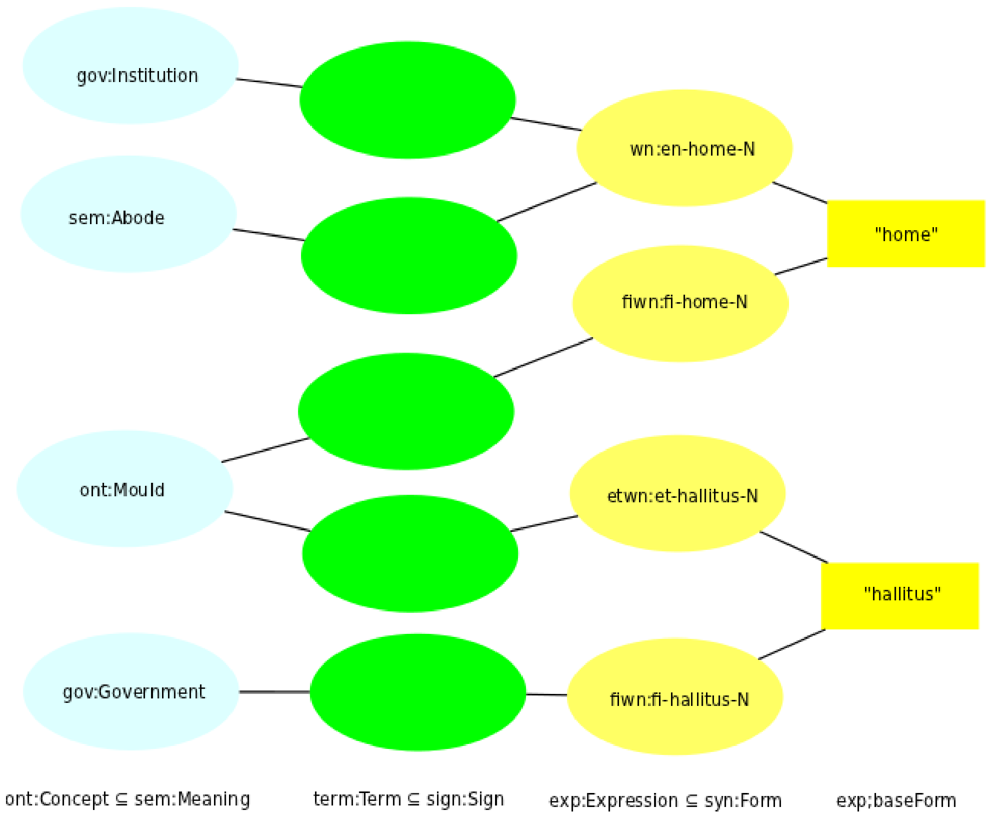


Figure 7: TermFactory top ontology schema

Noun example:

```
exp1:en—dog—N
   rdf:type owl:Thing , exp:Noun , exp:English ;
   exp:baseForm "dog" ;
   gf:cat gf:N ;
   gf:arguments [] .
```

```
term1:en-dog-N_-_ont-dog
   rdf:type owl:Thing , term:Term ;
   term:hasDesignation exp1:en-dog-N ;
   term:hasReferent ont0:Dog .
```

Adjective example:

```
exp1:en-divisible-A
   rdf:type exp:Adjective , exp:English ;
   exp:baseForm "divisible" ;
   gf:cat gf:A2 ;
   gf:arguments [
         gf:arg1 gf:by_Prep ] .

term1:en-divisible-A_-_wn30:synset-divisible-adjective-1
   rdf:type owl:Thing , term:Term ;
   term:hasDesignation exp1:en-divisible-A ;
   term:hasReferent wn30:synset-divisible-adjective-1 .
```

The format includes properties that are needed in GF grammars: GF category, gender, valency and the syntactic behaviour of the arguments. These are marked with the following predicates:

- `gf:cat`
- `gf:gender`
- `gf:arguments`
- `gf:arg{1,2,3...}`

The possible objects of `gf:cat` include the GF resource grammar library types, both argumentless, such as `N`, `CN` and `V`, and types that take arguments, such as `N3` (function *from* sth *to* sth), `A2` (divisible *by* sth) or `VV` (start *doing* sth). The predicate `gf:gender` is only for nouns, and not used in all languages.

The predicate `gf:arguments` takes a list, whose predicates are `gf:arg{1,2,3}` and each has as its object a value that tells the syntactic behaviour of the argument in question. The objects vary from language to language: for instance Bulgarian `gf:accusative` or `gf:dative`, English `gf:in_Prep` (believe *in* sth) or `gf:to_Prep` (want *to* do sth).

Some manual work is a prerequisite for an elaborate format such as this. Everything except the properties with prefix `gf:` belong to the standard TF format. Adding correct GF properties needs to be done manually, but when it is done, a conversion to GF format is straightforward. Currently the conversion program is a command-line tool, written in Java using the Jena RDF API, but it will be integrated to TF front end(s).

A user can add GF properties to TF entry, choosing from a pre-selected list, which properties are possible for a language in question. Seppo Nyrkkö‘s OntoR tool finds clusters of similar words, and its output is in the TF format with additional GF properties.

3.5 Related work

Very similar approach(to the one described in 3.2) for ontology verbalization with GF was pursuit in the scope of a European project of FP7 - PortDial[14]. Generation of GF grammars is based on a more elaborate core GF grammar, various templates, and "lemon"[15] - LEexicon Model of ONtologies.

The "lemon" framework is a product of another EU FP7 project - Monnet[16]. It provides in-depth lexical model that allows the easy incorporation of lexicons in different languages. Hence, it fascilitates the automatic creation of multilingual grammars.

3.6 Discussion on Levels of Automation

Verbalizing ontology data can be utilized by GF very successfully, but it requires human expertise in two directions:

- ontology expert - who will examine the domain ontologies and suggest generic SPARQL model for the domain;
- GF expert - who will implement a dynamic SPARQL model and the GF answer generation model.

Our experience has shown that there is constant need of synchronization between the two experts, with several iterations of refinement of the model.

In case we aim for verbalization of simple facts, the answer generation grammar can be generated in a semi automated manner, that consists of three major steps: ontology predicates analysis(automatic), GF pattern assignment(automatic, straightforward for English), and final GF expert edit.

[14]https://sites.google.com/site/portdial2
[15]http://lemon-model.net/
[16]http://www.monnet-project.eu/

4 Comparison and integration between KRI and Term Factory

As per the reviewers recommendation we have made comparison between KRI and TF and also we have suggested steps to integrate TF and KRI.

First, KRI is an information retrieval system over semantic repository. It allows RDF facts retrieval and their presentation in natural language, and optimally allows coverage in several languages. TermFactory is a higher-level system for management of term ontologies, a type of storage for storages. TF does not describe events or relations in the world. It maps terms to concepts, and focuses on multilinguality and the syntactic and morphologic completeness of the terms.

Both KRI and TermFactory make use of the Ontotext RDF databases in FactForge[17]. With KRI, the user can make queries to the databases in natural language. With TermFactory, the user can make the databases better suited for GF grammar and lexicon generation. When those collections are ported to a TermFactory site, there is a standard way to add grammatical properties to the expressions that do not have such properties yet. In addition, there is a standard way to convert from that representation format to GF grammars.

The information in FactForge is usually of the following type:

```
dbpedia:Salmon
  rdfs:label       "Salmon@en"
  dbp−ont:family   dbpedia:Salmonidae
  dbp−ont:class    dbpedia:Actinopterygii
```

All the properties about the resource `dbpedia:Salmon` are useful for querying. Once in TermFactory, the resource is linked to a linguistically rich expression in TF. For many of the common concepts there are already existing terms and expressions in some TermFactory repository, and all that needs to be done is to add a predicate that links `dbpedia:Salmon` to `exp1:en-salmon-N`, or for any other language, such as `exp1:sv-lax-N` for the Swedish expression for salmon. The latter resource would look like following:

```
exp1:en−salmon−N
   rdf:type owl:Thing , exp:Noun , exp:English ;
   exp:baseForm "salmon" ;
   gf:cat gf:N ;
   gf:arguments [] .
```

And, by the previous linking, the concept `dbpedia:Salmon` would be also linked to the following concept in TF top ontology.

```
term1:en−salmon−N_−_ont−salmon
   rdf:type owl:Thing , term:Term ;
   term:hasDesignation exp1:en−salmon−N ;
   term:hasReferent ont0:Salmon .
```

[17]http://factforge.net/

4.1 Integration

Integrating TF and KRI, steps to be taken:

1. (Setup) Ontotext has a TF site. We name it *OntoTF*.
2. (Setup) Ontotext has a large semantic repository, such as *FactForge*. We name it *OntoFF*.
3. *OntoTF* connects to a mirror instance of *OntoFF*, we name it *OntoFF_TF*. In *OntoFF_TF* editting is allowed.
4. Align the new semantic data with the current. Provide linkage through *owl:sameAs* predicate between, e.g. `dbpedia:Salmon` and `exp1:en-salmon-N`. `exp1:en-salmon-N` has a lot of additional properties in contrast to `dbpedia:Salmon`.
5. Querying *OntoFF_TF* will return TF enriched results

For example, if our query against *OntoFF* would return results such as `dbpedia:Dog`, the same query against *OntoFF_TF* would return the equivalent `term1:en-dog-N_-_ont-dog` and texttexp1:en-dog-N(details are listed below).

```
term1:en−dog−N_−_ont−dog
    rdf:type owl:Thing , term:Term ;
    rdfs:comment "an English term for the property ont0:Dog"^^xsd:string ;
    term:hasDesignation exp1:en−dog−N ;
    term:hasReferent ont0:Dog .

exp1:en−dog−N
    rdf:type owl:Thing , exp:Noun , exp:English ;
    rdfs:comment "an English noun"@en ;
    exp:baseForm "dog" ;
    gf:cat gf:N ;
    gf:arguments [] .
```

Furthermore, if the user performs a specific command, they will get generated GF lexicon such as:

```
−−This is a GF file produced automatically from a TermFactory lexicon.
−−# −path=.:../prelude:../abstract:../common
concrete TFDictEng of TFDict = CatEng ** open ParadigmsEng, ResEng, StructuralEng, Prelude in
     {
flags coding=utf8 ;

lin
  Dog_N = mkN "dog" ;
  Cat_N = mkN "cat" ;
  Horse_N = mkN "horse" ;
  ...
}
```

For terms that do not have a ready-made `expl:` entry in TF, some manual work is necessary. Valency and argument properties can be found from resources such as VerbNet[18] and FrameNet[19].

As for morphology, existing resources such as morphological lexicons should be exploited, as explained in [Lis12, pp. 54–57]. For instance, there is a complete mapping by Aarne Ranta from the classification system used by the Institute for Finnish Languages to the GF paradigms. For languages that do not have such resources, but have a GF resource grammar, the standard method could be created on base of GF. [Lis12, p. 55] describes a possible method:

> "A user interface would ask the user to write a basic form, typically nominative or infinitive, possibly using an example and an advice to write the entry word in the same form. The underlying GF web service would then run a `mkN`, `mkV` or similar and generate the whole paradigm. For highly inflecting languages it would show only some key forms. The user would correct what needs to be corrected and then accept the paradigm. In case there is nothing to correct, the way to construct that word in GF would be just `mk`*`C`* `<word>`. The smart paradigms in GF constructors have an order of what forms to include; for instance, a Finnish noun constructor with two forms takes the singular nominative and the plural partitive, so those would be the first forms to show to the user. When that is accepted, with or without correction, the user could be shown the next form, singular genitive. After 4 forms there is only the worst case constructor, 10 forms, and if all are accepted, then the paradigm of the word is saved to the entry."

However, this scenario has not been implemented. A `gf:paradigm` predicate can be added, but the values are highly language-dependent. Work in progress.

[18]http://verbs.colorado.edu/~mpalmer/projects/verbnet.html
[19]https://framenet.icsi.berkeley.edu/fndrupal/

5 MOLTO Prototypes

In this section we describe the MOLTO prototypes built entirely by Ontotext, or in which Ontotext participates. They are based on MOLTO KRI project, described in [MI10], [Dam11], [CEDR12] and also in the present document. The prototypes have both similarities and differences with respect to technologies used and grammar-ontology interoperability. The common technologies they make use of are:

- OWLIM 5([BKO+11]) - Ontotext's semantic repository
- Grammatical Framework for definition of query language, for the parsing of natural language queries and/or generation of natural language answers
- Forest 1.4. for user interface for information retrieval and browsing over semantic repository. (Forest currently is an internal software product of Ontotext, based on the Spring Framework[20])

In the following sections we provide more details on each particular prototype and explore the differences of the software's final version with a focus on grammar-ontology interoperability.

5.1 MOLTO KRI

The MOLTO KRI prototype is build as part of WP4 of MOLTO and is deployed at `http://molto.ontotext.com`.

5.1.1 Prototype Description

This prototype explores the PROTON[21] ontology which is focused on the types `Person`, `Location`, `Organization` and `Job Title`. It provides both querying and answers in natural languages, making use of GF(examples on Figures 8 and 9). Query to SPARQL is done via the mapping rules module(Section 2.1) and RDF to natural language follows the approach of verbalizing predicates with semi-auto-generated grammars(Section 3).

5.1.2 Grammar-Ontology Interoperability

The query grammar was designed by Ontotext members and created by experts from UGOT, in 6 different languages. (We added also Bulgarian query grammar for completeness.) The grammars can be found at `svn://molto-project.eu/wp4/projects/molto-kri/natural-language-queries/resources/grammars/query/`. The natural language queries are parsed to GF abstract representation, which is mapped to SPARQL

[20]http://www.springsource.org/spring-framework

[21]`http://www.ontotext.com/proton-ontology`

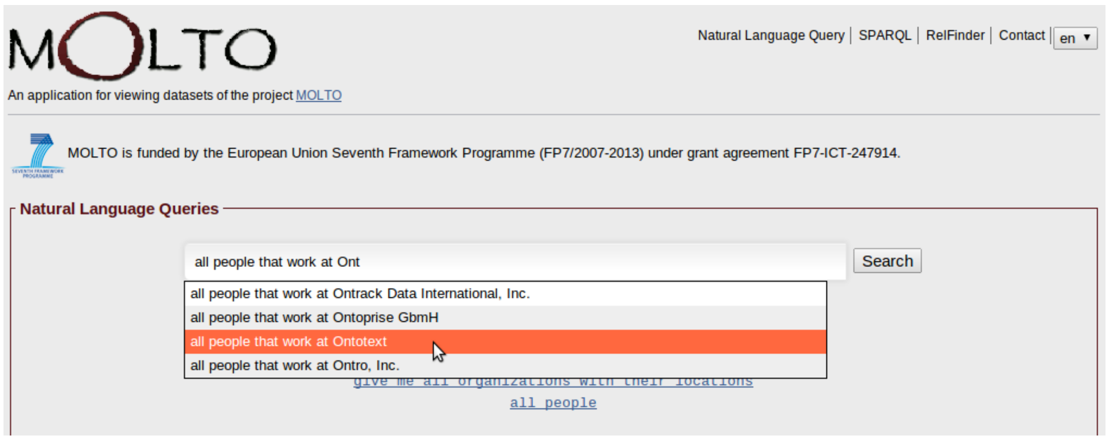
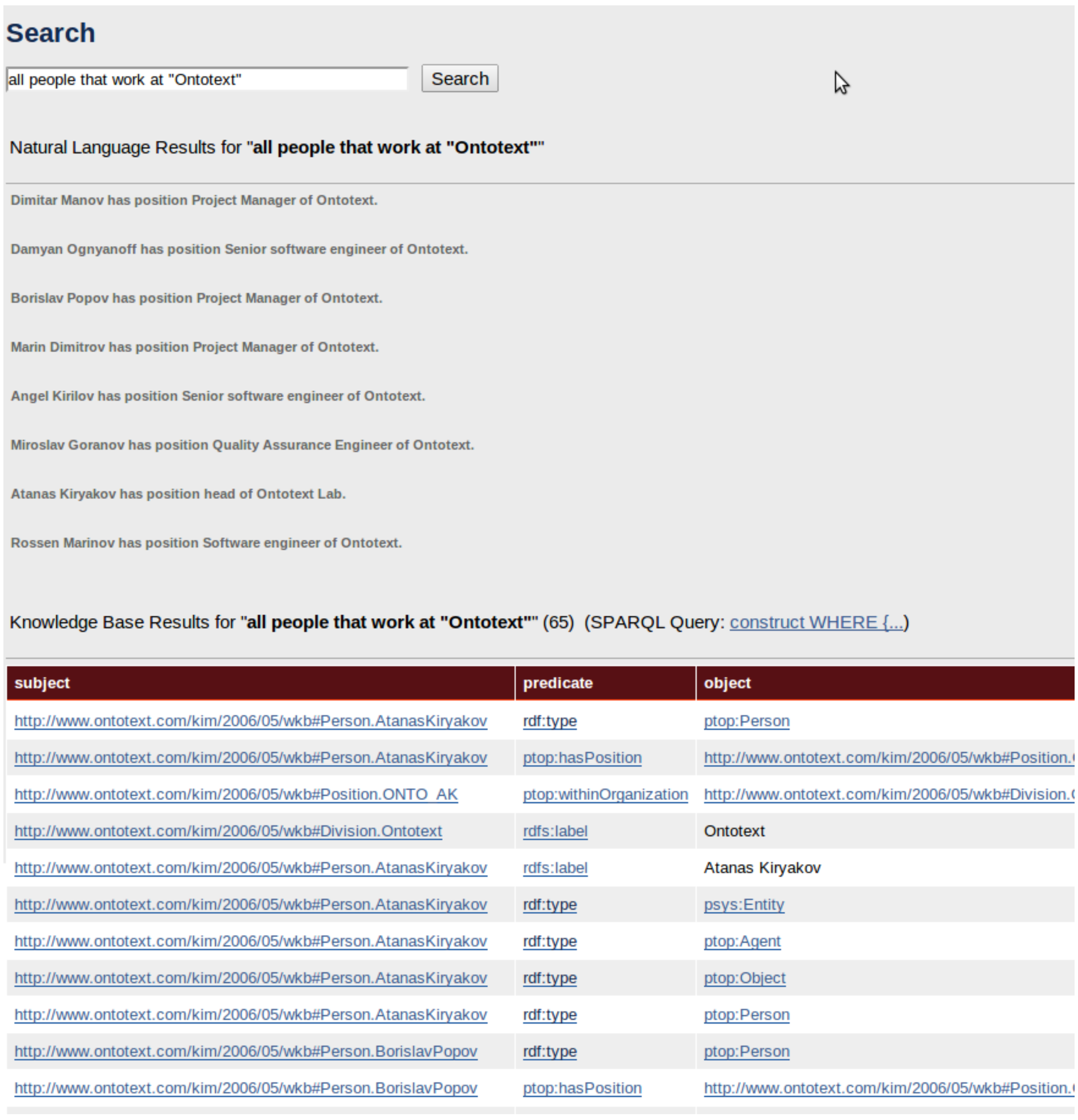

MOLTO
Natural Language Query | SPARQL | RelFinder | Contact | en
An application for viewing datasets of the project MOLTO
MOLTO is funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement FP7-ICT-247914.
Natural Language Queries
all people that work at Ont
Search
all people that work at Ontrack Data International, Inc.
all people that work at Ontoprise GbmH
all people that work at Ontotext
all people that work at Ontro, Inc.
give me all organizations with their locations
all people

Figure 8: MOLTO KRI queries

queries via mapping rules(Section 3.2). We kept them just to demonstrate a second approach, but we officially quitted this technologies in the MOLTO KRI child projects.

The SPARQL queries are “construct” queries, which return RDF graph, that consists of “subject-predicate-objects” triples. Our task was the verbalization of these triples, with focus on the predicates. To resolve it, we use separate module of the answer grammar. Originally, it was semi-automatically generated, and in English only. In the final year of MOLTO it was refactored and other answer languages were added by the UGOT experts.

The whole answer grammar has two main parts - data and expressions(based on RDF predicates). The data grammar was automatically generated from the ontology. It contains over 135000 entities from specific classed selected for the purpose of the prototype. They were extracted from the ontology and presented in GF under the general `Entity` category. Each subject or object is described by the meaningful part of their URI(the one without the domain) and the English RDF label from the ontology. The resulting abstract GF representation is demonstrated on Section 3.2.1 below. The GF grammars for verbalization of predicates are semi-automatically generated and manually translated to the different languages.



Search

all people that work at "Ontotext"

Natural Language Results for "**all people that work at "Ontotext"**"

Dimitar Manov has position Project Manager of Ontotext.

Damyan Ognyanoff has position Senior software engineer of Ontotext.

Borislav Popov has position Project Manager of Ontotext.

Marin Dimitrov has position Project Manager of Ontotext.

Angel Kirilov has position Senior software engineer of Ontotext.

Miroslav Goranov has position Quality Assurance Engineer of Ontotext.

Atanas Kiryakov has position head of Ontotext Lab.

Rossen Marinov has position Software engineer of Ontotext.

Knowledge Base Results for "**all people that work at "Ontotext"**" (65) (SPARQL Query: construct WHERE {...})

subject	predicate	object
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdf:type	ptop:Person
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	ptop:hasPosition	http://www.ontotext.com/kim/2006/05/wkb#Position.
http://www.ontotext.com/kim/2006/05/wkb#Position.ONTO_AK	ptop:withinOrganization	http://www.ontotext.com/kim/2006/05/wkb#Division.
http://www.ontotext.com/kim/2006/05/wkb#Division.Ontotext	rdfs:label	Ontotext
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdfs:label	Atanas Kiryakov
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdf:type	psys:Entity
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdf:type	ptop:Agent
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdf:type	ptop:Object
http://www.ontotext.com/kim/2006/05/wkb#Person.AtanasKiryakov	rdf:type	ptop:Person
http://www.ontotext.com/kim/2006/05/wkb#Person.BorislavPopov	rdf:type	ptop:Person
http://www.ontotext.com/kim/2006/05/wkb#Person.BorislavPopov	ptop:hasPosition	http://www.ontotext.com/kim/2006/05/wkb#Position.

Figure 9: MOLTO KRI results

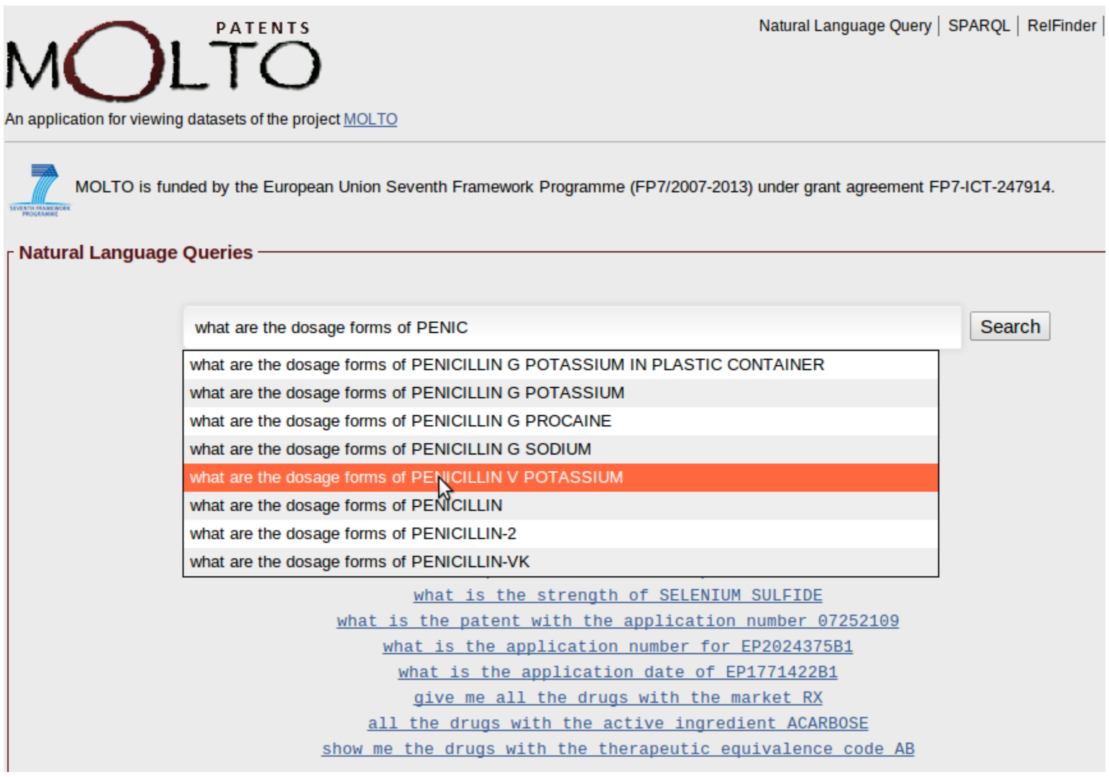
5.2 MOLTO Patents

MOLTO Patents was build as part of WP7 of MOLTO and is a child project of MOLTO KRI. A public prototype can be found at `http://molto-patents.ontotext.com`.

5.2.1 Prototype Description

This prototype allows natural language queries against semantically annotated biomedical patents from the EPO[22] corpus and the FDA[23] patents ontology, aligned with other applicable ones. It is a document and RDF retrieval system that allows queries in three natural languages - English, French and German. The query grammar was significantly improved for the current version of the prototype. It was written by GF experts from UGOT.

In this use case we did not apply NL answers. Instead, we retrieve ontology data and biomedical patents, translated with statistical machine translation techniques by UPC.



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.

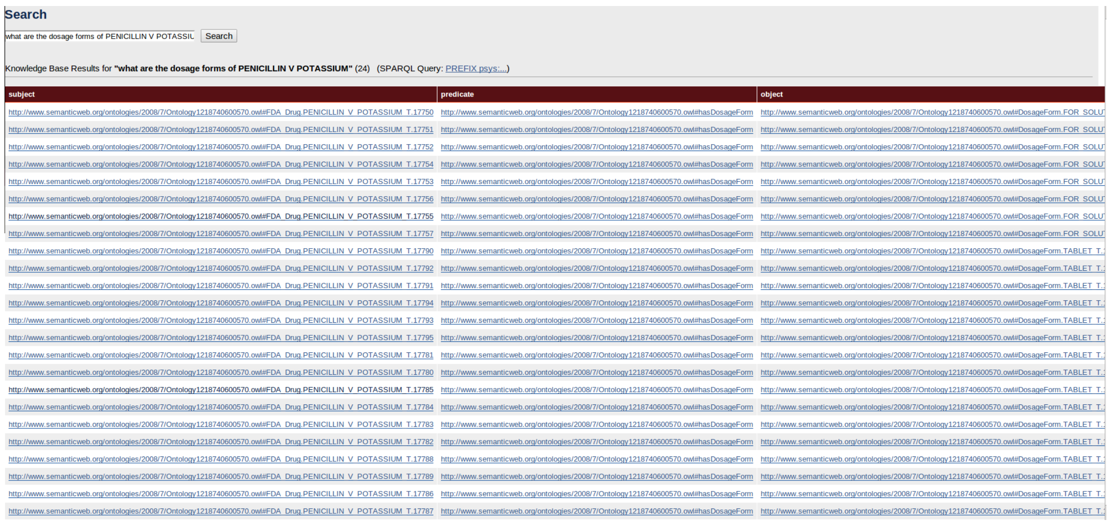
what are the dosage forms of PENIC

what are the dosage forms of PENICILLIN G POTASSIUM IN PLASTIC CONTAINER
what are the dosage forms of PENICILLIN G POTASSIUM
what are the dosage forms of PENICILLIN G PROCAINE
what are the dosage forms of PENICILLIN G SODIUM
what are the dosage forms of PENICILLIN V POTASSIUM
what are the dosage forms of PENICILLIN
what are the dosage forms of PENICILLIN-2
what are the dosage forms of PENICILLIN-VK

what is the strength of SELENIUM SULFIDE
what is the patent with the application number 07252109
what is the application number for EP2024375B1
what is the application date of EP1771422B1
give me all the drugs with the market RX
all the drugs with the active ingredient ACARBOSE
show me the drugs with the therapeutic equivalence code AB

Figure 10: MOLTO patents queries

[22]European Patents Office
[23]Food and Drug Administration Agency, US



Search

what are the dosage forms of PENICILLIN V POTASSIU Search

Knowledge Base Results for **"what are the dosage forms of PENICILLIN V POTASSIUM"** (24) (SPARQL Query: PREFIX psys:...)

subject	predicate	object
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17750	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17751	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17752	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17754	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17753	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17756	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17755	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17757	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.FOR_SOLU
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17790	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17792	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17791	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17794	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17793	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17795	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17781	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17780	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17785	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17784	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17783	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17782	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17788	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17789	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17786	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.
http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#FDA_Drug.PENICILLIN_V_POTASSIUM_T.17787	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#hasDosageForm	http://www.semanticweb.org/ontologies/2008/7/Ontology1218740600570.owl#DosageForm.TABLET_T.

Figure 11: MOLTO patents results

5.2.2 Grammar-Ontology Interoperability

In MOLTO patents we applied for the first time the GF generating SPARQL approach, defined in Section 2.2.2. We had the chance to explore its advantages over the mapping rules which were permanently substituted. It is interesting to point out that the query language was based on ontology relations(e.g. “what are the active ingredients of BACLOFEN”) and also on semantic annotations over the documents(e.g. “patents that mention PENICILLIN and AMPICILLIN”) where the results are documents that were annotated to contain both drugs. In general, the query language was based on ontology relations and data, and on semantic data that we added through GATE[24] annotation pipeline.

[24]http://gate.ac.uk/

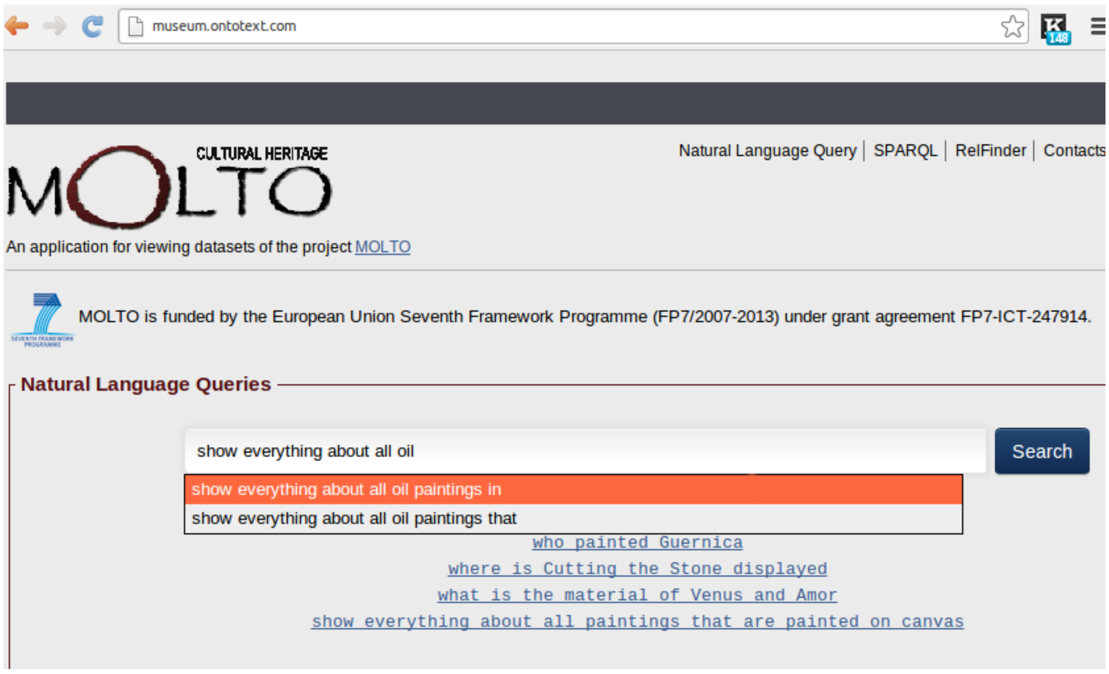
5.3 MOLTO Cultural Heritage

MOLTO Cultural Heritage was build as part of WP8 of MOLTO and is a child project of MOLTO KRI as well. The public prototype is located at `http://museum.ontotext.com`.

5.3.1 Prototype Description

An aligned model of ontologies in the cultural heritage domain provides data for retrieval system, that generates natural language descriptions of museum paintings objects. This prototype provides queries on the topic of "museum paintings" in natural language in 15 languages.

Query interface is shown on Figure 12.



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

show everything about all oil

show everything about all oil paintings in

show everything about all oil paintings that

who painted Guernica

where is Cutting the Stone displayed

what is the material of Venus and Amor

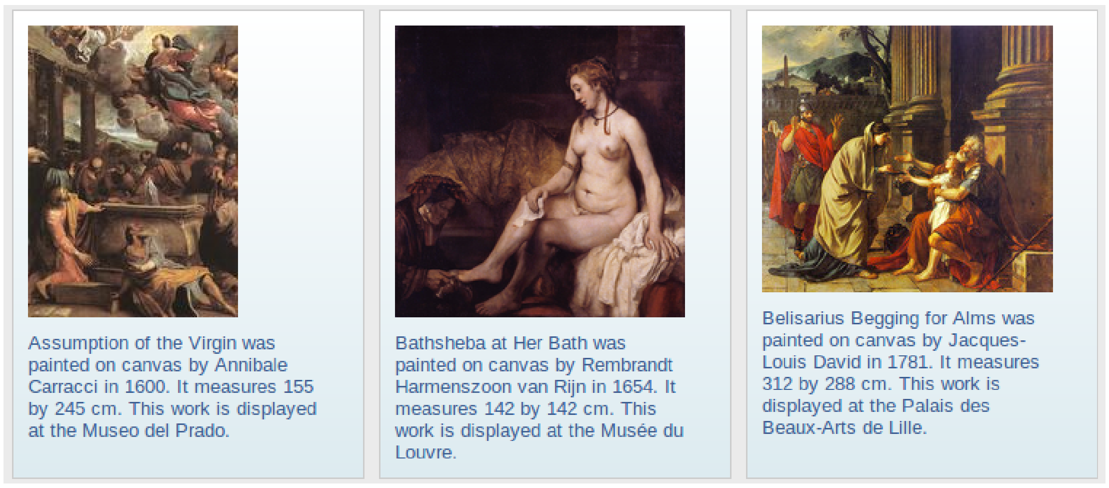
show everything about all paintings that are painted on canvas

Figure 12: MOLTO cultural heritage queries

The corresponding result(truncated version) is displayed on Figure 13.

5.3.2 Grammar-Ontology Interoperability

The query language is designed to cooperate with the `painting.owl` ontology, to which all other paintings data was mapped. In this use case the focus was on generaton of painting description. Automatically generated grammar was not applicable. Anyway, data dictionaries for the grammars were provided partially from the ontology data in the available languages. We used "select" queries to retrieve specific painting details and pass them to a GF function that generated painting description; details in Section 3.3.2. For GF-to-SPARQL we directly used the new approach described in in Section 2.2.

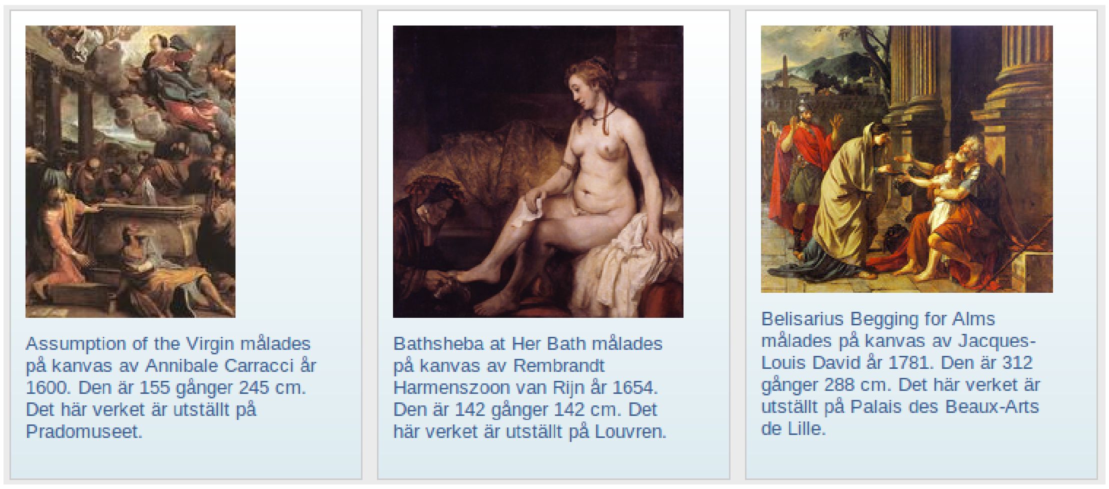


Assumption of the Virgin was painted on canvas by Annibale Carracci in 1600. It measures 155 by 245 cm. This work is displayed at the Museo del Prado.

Bathsheba at Her Bath was painted on canvas by Rembrandt Harmenszoon van Rijn in 1654. It measures 142 by 142 cm. This work is displayed at the Musée du Louvre.

Belisarius Begging for Alms was painted on canvas by Jacques-Louis David in 1781. It measures 312 by 288 cm. This work is displayed at the Palais des Beaux-Arts de Lille.

Figure 13: MOLTO cultural heritage results - paintings descriptions, English



Assumption of the Virgin målades på kanvas av Annibale Carracci år 1600. Den är 155 gånger 245 cm. Det här verket är utställt på Pradomuseet.

Bathsheba at Her Bath målades på kanvas av Rembrandt Harmenszoon van Rijn år 1654. Den är 142 gånger 142 cm. Det här verket är utställt på Louvren.

Belisarius Begging for Alms målades på kanvas av Jacques-Louis David år 1781. Den är 312 gånger 288 cm. Det här verket är utställt på Palais des Beaux-Arts de Lille.

Figure 14: MOLTO cultural heritage results - paintings descriptions, Swedish

6 Porting KRI to New Applications

In this section we describe porting of KRI to new use cases, based on the experience with the prototypes we built.

Hereby we present the list of steps needed to adapt KRI for a new domain. Some of the steps are optional, depending on the task solved and on the initial resources for it.

1. Examine the domain. Align semantic data and ontologies, if needed.
2. Suggest applicable query language and create query grammars(either manually, or extract them from the ontology).
3. Define "variable" types - for GF data grammars and for the data lexicons for the FSA autocomplete.
4. Create exemplary SPARQL queries for the GF model. A SPARQL expert is required.
5. Create dynamical model for the domain specific SPARQL generation. A SPARQL expert is required.
6. Create/correct manually the GF grammars for query and/or answer. This process may require a few iterations. The experts previously included should cooperate.
7. Translate the query grammars in other natural languges.
8. Generate autocomplete resources from the produced grammars.
9. Add the grammars to the prototype and suggest example queries for the UI.
10. Test the resulting system. If not satisfied, return to step 6.

The process of refining the GF SPARQL grammar usually included a few iterations, where we were changing the natural language queries and the expected result.

6.1 On the strengths and limitations of our natural language interface to semantic repositories

A grammar-based interface from natural language queries to ontologies ensures maximum accuracy of the results that the user receives. The natural language query is translated into SPARQL without ambiguities, ensuring the correctness of the triples returned. In domain-specific applications, this is a highly appreciated quality. For example, when searching information in Patents (see our demo), or in other types of legal documents, imprecise, missing or wrong results may have seriously negative consequences.

However, in order to achieve maximum accuracy, the MOLTO interface suffers from a couple of limitations. First, usually a predefined list of NL queries needs to be created, commonly by experts in the specific domain of the aplication. They reflect the information most frequently required by users from the knowledge base. Second, a GF expert needs to implement the query grammar that can generate all predefined queries. The grammar has an abstract core and several concrete grammars, if multiple language support is required by the application. A special concrete grammar offers immediate translation to SPARQL queries, the key ingredient that lays at the base of the grammar-ontology interoperability. Especially for the SPARQL concrete grammar, a SPARQL expert is needed. Third, an GF answer grammar needs to implemented, in order to process the triples that result after running the SPARQL query into natural language answers. This step may not be necessary sometimes.

Regarding the last step, we suggest already in this appendix some more automated way of verbalizing RDF triples, that can facilitate the automatization of the answer grammar.

7 Conclusion

Throughout MOLTO we have explored a few different directions of grammar-ontology interoperability. Our experience has shown that Grammatical Framework is suitable as a means for generation of grammars that define a query language, including corresponding machine language that to be used for query execution against the retrieval system. Verbalization of response is easily supported as well. It can be focused on only the simple facts verbalization, or it could be used for parametrized descriptions, which we both demonstrarted in the related work packages. The degree of automation of the whole interoperability cycle can be very high, but this requires high expertise from GF point of view, good understanding of the domain, and good planning of the verbalization model.

8 References

References

[AGL13] Alessio Palmero Aprosio, Claudio Giuliano, and Alberto Lavelli. *Automatic Expansion of DBpedia Exploiting Wikipedia Cross-Language Information.* Montpellier, France, May 2013. ESWC.

[BKO$^+$11] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web Journal*, 2:33–42, June 2011.

[CEDR12] Milen Chechev, Ramona Enache, Mariana Damova, and Aarne Ranta. *D4.3 Grammar-Ontology Interoperability*, May 2012. Deliverable 4.3. MOLTO FP7-ICT-247914.

[Con11] MOLTO Consortium. Molto enlarged eu annex i - description of work, 2011.

[Dam11] Mariana Damova. *D4.2 Data Models and Alignment*, May 2011. Deliverable 4.2. MOLTO FP7-ICT-247914.

[DDE$^+$13] Mariana Damova, Dana Dannells, Ramona Enache, Maria Mateva, and Aarne Ranta. Natural language interaction with semantic web knowledge bases and lod. In Paul Buitelaar and Philip Cimiano, editors, *Towards the Multilingual Semantic Web*. Springer, Heidelberg, Germany, 2013.

[DRE$^+$13] Dana Dannells, Aarne Ranta, Ramona Enache, Mariana Damova, and Maria Mateva. *D8.3 Translation and retrieval system for museum object descriptions*, 2013. Deliverable 8.3. MOLTO FP7-ICT-247914.

[Ena10] Ramona Enache. Reasoning and language generation in the sumo ontology. Master's thesis, Chalmers University of Technology, 2010.

[Lis12] Inari Listenmaa. Ontology-based lexicon management in a multilingual translation system – a survey of use cases. Master's thesis, University of Helsinki, November 2012.

[MGE$^+$13] Maria Mateva, Meritxell Gonzàlez, Ramona Enache, Cristina España-Bonet, Lluís Màrquez, Borislav Popov, and Aarne Ranta. *D7.3 Patent MT and Retrieval. Final Report.*, April 2013. Deliverable 7.3. MOLTO FP7-ICT-247914.

[MI10] Peter Mitankin and Atanas Ilchev. *D4.1 Knowledge Representation Infrastructure*, November 2010. Deliverable 4.1. MOLTO FP7-ICT-247914.

[MS06] Chris Mellish and Xiantang Sun. The semantic web as a linguistic resource: opportunities for natural language generation. knowledge based systems. In *Presented at the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 2006.

[Ran12] A. Ranta. *Implementing Programming Languages. An Introduction to Compilers and Interpreters, with an appendix coauthored by Markus Forsberg.* College Publications, London, 2012.

9 Appendix A: Groups of RDF predicate types

Type Name	Pattern	Examples	# of pred.
TYPE A0	GENERAL A Group type	-	11
TYPE A1	E1 has X E2.	has president, has profession	533
TYPE A2	E1 has X X E2.	has subsequent work has general manager	643
TYPE A3	E1 has X Prep E2.	has route of administration has distance to london	53
TYPE A4	E1 has X X X E2.	has free flight time has first driver team	179
TYPE A5	E1 has X Prep X X E2.	has distance to charing cross has number of doctoral students	17
TYPE A6	E1 has X X Prep X E2.	has end year of insertion has ethnic groups in year	8
TYPE A7	E1 has X X X X E2.	has port 1 docking date has original maximum boat beam	30
TYPE A8	E1 has X Prep X X X E2.	has rank in final medal count has percentage of area water round	6
TYPE A9	E1 has X X Prep X X E2.	has qmf language of short tmpl has qmf strsqlval of short tmpl	18
TYPE A10	E1 has X X X Prep X E2.	has qmf is subformat of long	1
TYPE A11	E1 has X X X X X E2.	has national topographic system map number	8
TYPE B0	GENERAL B Group type	-	8
TYPE B1	E1 is X Prep E2.	is similar to is complement of	87
TYPE B2	E1 is X X Prep E2.	is doing business as is child organization of	27
TYPE B3	E1 is Prep E2.	is from is within	4
TYPE C0	GENERAL C Group type	-	6
TYPE C1	E1 Verb E2.	contains permits	59
TYPE C3	E1 Verb Prep E2.	conforms to refers to	11
TYPE C4	E1 Verb Prep X E2.	wins at asia wins at challenges	13
TYPE C5	E1 Verb X Prep E2.	was intended for shows features of	2

Table 2: RDF predicates types classification with respect ot GF patterns