CLOVA: An Architecture for Cross-Language Semantic Data Querying

John McCrae
Semantic Computing Group, CITEC
University of Bielefeld
Bielefeld, Germany
jmccrae@cit-ec.uni-bielefeld.de

Jesús R. Campaña
Department of Computer Science and Artificial Intelligence
University of Granada
Granada, Spain
jesuscg@decsai.ugr.es

Philipp Cimiano
Semantic Computing Group, CITEC
University of Bielefeld
Bielefeld, Germany
cimiano@cit-ec.uni-bielefeld.de

ABSTRACT

Semantic web data formalisms such as RDF and OWL allow us to represent data in a language independent manner. However, so far there is no principled approach allowing us to query such data in multiple languages. We present CLOVA, an architecture for cross-lingual querying that aims to address this gap. In CLOVA, we make a distinction between a language independent *data layer* and a language independent *lexical layer*. We show how this distinction allows us to create modular and extensible cross lingual applications that need to access semantic data. We specify the search interface at a conceptual level using what we call a *semantic form specification* abstracting from specific languages. We show how, on the basis of this conceptual specification, both the query interface and the query results can be localized to any supported language with almost no effort. More generally, we describe how the separation of the lexical layer can be used with a principled ontology lexicon model (LexInfo) in order to produce application-specific lexicalisations of properties, classes and individuals contained in the data.

Categories and Subject Descriptors

H.5.m [**Information Interfaces and Presentation**]: User Interfaces; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; I.2.7 [**Artificial Intelligence**]: Natural Language Processing

General Terms

Design, Human Factors, Languages

Keywords

Multilingual Semantic Web, Ontology Localisation, Software Architecture

1. INTRODUCTION

Data models and knowledge representation formalisms in the Semantic Web allow us to represent data without reference to natural language[1]. In order to facilitate the interaction of human users with semantic data, supporting language-based interfaces in multiple languages is crucial. However, currently there is no principled approach supporting the access of semantic data across multiple languages. To fill this gap, we present in this paper an architecture we call CLOVA (Cross-Lingual Ontology Visualisation Architecture) designed for querying semantic data in multiple languages. A developer of a CLOVA application can define the search interface independently of any natural language by referring to ontological relations and classes within a *semantic form specification* (SFS), which represents a declarative and conceptual representation of the search interface with respect to a given ontology. We have designed an XML-based language which is inspired by the Fresnel language [2] for this purpose. The search interface can then be automatically localised by the use of a lexicon ontology model such as LexInfo [4], enabling the system to automatically generate the form in the appropriate language. The queries to the semantic repository are generated on the basis of the information provided in the SFS and the results of the query can be localised using the same method as used for the localisation of the search interface. The CLOVA framework is generic in the sense that it can be quickly customised to new scenarios, new ontologies and search forms and additional languages can be added without changing the actual application, even at run time if we desire.

The paper is organised as follows. Section 2 describes state of the art on information access across languages and points out basic requirements for cross lingual systems. Section 3 describes the CLOVA framework for rapid development of cross-lingual search applications accessing semantic data. We conclude in Section 4.

2. RELATED WORK

Providing access to information across languages is an important topic in a number of research fields. While our work is positioned in the area of the Semantic Web, we discuss work related to a number of other research areas, including databases, cross-language information retrieval as well as ontology presentation and visualisation.

WWW2010, April 26-30, 2010, Raleigh, North Carolina.
.

[1] This holds mainly for RDF triples with resources as subjects and objects. String data-type elements are often language-specific.

2.1 Database Systems

Supporting cross-language data access is an important topic in the area of database systems, albeit one which has not received very prominent attention (see [8]). An important issue is certainly the one of character encoding as we need to represent characters for different languages. However, most of the current database systems support Unicode so that this issue is not a problem anymore. A more complex issue is the representation of content in the database in such a way that information can be accessed across languages. There seems to be no consensus so far on what the optimal representation of information would be such that cross-language access can be realised effectively and efficiently. One of the basic requirements for multilingual organisation of data mentioned by Kumaran et al. [8] is the following:

"The basic multilingual requirement is that the database system must be capable of storing data in multiple languages."

This requirement seems definitely too strict to us as it assumes that the representation of data is language-dependent and that the database is supposed to store the data in multiple languages. This rules out language-independent approaches which do not represent language-specific information in the database at all.
The following requirement by Kumaran et al. is one we can directly adhere to:

Requirement 1 (Querying in multiple languages)
Data must be queriable using query strings in any (supported) language.

In fact, we will refer to the above as *Requirement 1a* and add the following closely related *Requirement 1b*: *'The results of a query should also be presented in any (supported) language.'* Figure 1 summarises all the requirements discussed in this section. However, it does not strictly follow from this that the data should be stored in multiple languages in the database. In fact, it suffices that the front end that users interact with supports different languages and is able to translate the user's input into a formal (language-independent) query and localises the results returned by the database management system (DBMS) into any of the supported languages.

A further important requirement by Kumaran et al. we subscribe to is related to interoperability:

Requirement 2 (Interoperability)
The multilingual data must be represented in such a way that it can be exchanged across systems.

This feature is certainly desirable. We will come back to this requirement in the context of our discussion of the Semantic Web (see below). The next two requirements mentioned by Kumaran et al. are in our view questionable as they assume that the DBMS itself has built-in support for multiple languages:

- **String equality across scripts**: A Multilingual database system should support *lexical joins* allowing to join information in different tables even if the relevant attributes of the join are in different scripts.
- **Linguistic equivalences**: Multilingual database systems should support *linguistic joins* which exploit predefined mappings between attributes and values across languages. For example, we might state explicitly that the attributes *"marital status"* (in English) and *"Familienstand"* are equivalent and that the values *"married"* and *"verheiratet"* are equivalent.

In fact, those two requirements follow from Kumaran et al.'s assumption that the database should store the data in multiple languages. If this is the case then we certainly have to push all the cross-language querying functionality into the DBMS itself. This is rather undesirable from our point of view as every time a new language is added to the system, the DBMS needs to be modified to extend the linguistic and lexical equivalences. Further, the data is stored redundantly (once for every language supported). Therefore, we actually advocate a system design where the data is stored in a language-independent fashion and the cross-lingual querying functionality as well as result localisation is external to the DBMS itself, implemented as pre- and post-processing steps, respectively.

In fact, we would add the following requirement to any system allowing to access data across languages:

Requirement 3 (Language Modularity)
The addition of further languages should be modular in the sense that it should not require the modification of the DBMS or influence the other languages supported by the system.

As a consequence, the capability of querying data across languages should not be specific to a certain implementation of a DBMS but work for any DBMS supporting the data model in question.

One of the important issues in representing information in multiple languages is avoiding redundancy (see [6]). Hoque et al. indeed propose a schema to give IDs to every piece of information and then include the language information in a dictionary table. This is perfectly in line with Semantic Web data models (RDF in particular) where URIs are used to uniquely identify resources. Dictionaries can then be constructed expressing how the elements represented by the URIs are referred to across languages. This thus allows to conceptually separate the data from the dictionary. This is a crucial distinction that CLOVA also adheres to (see below).

2.2 Cross-language Information Retrieval

In the field of information retrieval, information access across languages has also been an important topic, mainly in the context of the so called *Cross-Language Evaluation Forum*[2] (see [10] for the proceedings of CLEF 2008). Cross-language information retrieval (CLIR) represents an extreme case of the so called vocabulary mismatch problem well-known from information retrieval. The problem, in short, is the fact that a document can be highly relevant to a query in spite of not having any words in common with the query. CLIR represents an extreme case in the sense that if a query and a document are in different languages, then the word overlap and consequently every vector-based similarity measure will be zero.

[2] http://www.clef-campaign.org/

In CLIR, the retrieval unit is the document, while in database systems the retrieval unit corresponds to the information units stored in the data base. Therefore, the requirements with respect to multilinguality are rather different for CLIR and multilingual database systems.

2.3 Semantic Web

Multilinguality has been so far an underrepresented topic in the Semantic Web field. While on the Semantic Web we encounter similar problems as in the case of databases, there are some special considerations and requirements. We will consider further important requirements for multilinguality in the context of the Semantic Web. Before, we introduce the crucial distinction between the **data layer (proper)** and the **lexical layer**. We will see below that the conceptual separation between the data and the dictionary is even more important in the context of the Semantic Web. According to our distinction, the data layer contains the application-relevant data while the lexical layer merely contains information about how the data is realised/expressed in different languages and acts like a dictionary. We note that this distinction is a conceptual one as the data in both layers can be stored in the same DBMS. However, this might not always be possible in a decentralised system such as the Semantic Web:

Requirement 4 (Data and Lexicon Separation)
We require a clear separation between the data and lexicon layer in the Semantic Web. The addition of further languages should be possible without modifying the data layer. This means that the proper data layer and the lexical layer are cleanly separated and data is not stored redundantly.

In the Semantic Web, the parties interested in accessing a certain data source are not necessarily its owners (in contrast to standard centralised database systems as considered by Kumaran et al.). As a corollary it follows that if a user requires access to a data source in language x he might not have the permission to enrich the data source by data represented in the language x.

A further relevant requirement in the context of the Semantic Web is the following:

Requirement 5 (Sharing of Lexica)
Lexica should be represented declaratively and in a form which is independent of specific applications such that it can be shared.

It is very much in the spirit of the Semantic Web that information should be interoperable and thus reusable beyond specific applications. Following this spirit, it seems desireable that (given that data representation is language-independent) the language-specific information how certain resources are expressed in various languages can be shared across systems. This can be accomplished by declaratively described lexica which can be shared.

Multilinguality has been approached in RDF through the use of its `label` property, which can assign labels with language annotations to URIs. The SKOS framework [9] further expands on this by use of `prefLabel`, `altLabel`, `hiddenLabel`. These formalisms are sufficient for providing simple representation of language information. However, as more complex lexico-syntactic information is required, in turn more complex representations are necessary. A more formal distinction of the "data layer" and "lexical layer" is provided by *lexicon ontology models* of which the most prominent models are the Linguistic Information Repository (LIR) and LexInfo (see [4]).

2.4 Ontology Presentation and Visualisation

Fresnel [2] is a display vocabulary that describes methods of data presentation in terms of *lenses* and *formats*. In essence the *lens* in Fresnel selects which values are to be displayed and the *format* selects the formatting applied to each part of the lens. This provides many of the basic tools for presenting semantic web data. However it does not represent multilinguality within the vocabulary and it is not designed to present a queriable interface to the data. There exist many forms of ontology visualisation methods through the use of trees, and other structures to display the data contained within the ontology, a survey of which is provided in [7]. These are of course focussed mainly on displaying the structure of the ontology and do not attempt to convert the ontology to natural language. Furthermore, for very large data sources, it is impractical to visualise the whole ontology at one time and hence we wish only to select a certain section of it and hence require a query interface to perform this task.

3. MULTILINGUAL ACCESS AND QUERYING USING CLOVA

CLOVA addresses the problem of realising localised search interfaces on top of language-independent data sources, abstracting the work flow and design of a search engine and providing the developer with a set of tools to define and develop a new system with relatively little effort. CLOVA abstracts lexicalisation and data storage as services, providing a certain degree of independence from data sources and multilingual representation models.

The different modules of the system have been designed with the goal of providing very specific, non-overlapping and independent tasks to developers working on the system deployment concurrently. User interface definition tasks are completely separated from data access and lexicalisation, allowing developers of each module to use different resources as required.

CLOVA as an architecture does not fulfil any of the aforementioned requirements (as they should be fulfilled by lexicalisation services), but provides a framework to fully exploit cross-lingual services meeting these requirements. The application design allows to separate conceptual representations from language dependant lexical representations, making user interfaces completely language independent in order to later localise them to any supported language.

3.1 System Architecture

The CLOVA architecture is designed to enable the querying of semantic data in a language of choice, while still presenting queries to the data source in a language-independent form. CLOVA is modular, reusable and extensible and as such is easily configured to adapt to different data sources, user interfaces and localisation tools[3].

[3] A Java implementation of CLOVA is available at `http://www.sc.cit-ec.uni-bielefeld.de/clova/`

Req. No	Implication	Status
Req. 1a	Querying in multiple languages	REQUIRED
Req. 1b	Result localisation in multiple languages	REQUIRED
Req. 2	Data interoperability	REQUIRED
Req. 3	Language modularity	REQUIRED
Req. 4a	Separation between data and lexical layer	DESIRED TO SUPPORT Req. 3
Req. 4b	Language-independent data representation	DESIRED TO AVOID REDUNDANCY
Req. 5	Declarative representation of lexica	DESIRED FOR SHARING LEXICAL INFORMATION

Figure 1: Requirements for multilingual organisation of data

Figure 2 depicts the general architecture of CLOVA and its main modules. The *form displayer* is a module which translates the semantic form specification into a displayable format, for example HTML. Queries are performed by the *query manager* and then the results are displayed to the user using the *output displayer* module. All of the modules use the *lexicaliser* module to convert the conceptual descriptions (i.e., URIs) to and from natural language. Each of these modules are implemented independently and can be exchanged or modified without affecting the other parts of the system.

We assume that we have a data source consisting of a set of properties referenced by URIs and whose values are also URIs or language-independent data values. We shall also assume that there are known labels for each such URI and each language supported by the application. If this separation between the lexical layer and the data layer does not already exist, we introduce elements to create this separation. It is often necessary to apply such manual enrichment to a data source, as it is not trivial to identify which strings in the data source are language-dependent, however we find that is often a simple task to perform by identifying which properties have language-dependent ranges, or by using XML's language attribute.

We introduce an abstract description of a search interface by way of XML called a *semantic form specification.* It specifies the relevant properties that can be queried by using the URIs in the data source, thus abstracting from any natural language. We show how this can be used to display a form to the user and to generate appropriate queries once he/she has filled in the form. The *query manager* provides a backend that allows us to convert our queries using information in the form into standard query languages such as SPARQL and SQL. Finally, we introduce a lexicalisation component, which is used to translate between the language-independent forms specified by the developer and the localised forms presented to the user. We describe a lexicaliser which builds on a complex lexicon model and demonstrate that it can provide more flexibility with respect to the context and complexity of the results we wish to lexicalise.

3.2 Modules

3.2.1 Semantic Form Specification

One of the most important aspects of the architecture is the *Semantic Form Specification* (SFS), which contains all the necessary information to build a user interface to query the ontology. In the SFS the developer specifies the ontology properties to be queried by the application via their URIs. This consists of a form for which we specify a *domain*, i.e., the class of objects we are querying as defined in the database by an RDF `type` declaration or similar. If this is omitted we simply choose all individuals in the data source. The SFS essentially consists of a list of fields which are to be used to query the ontology. Each field contains the following information:

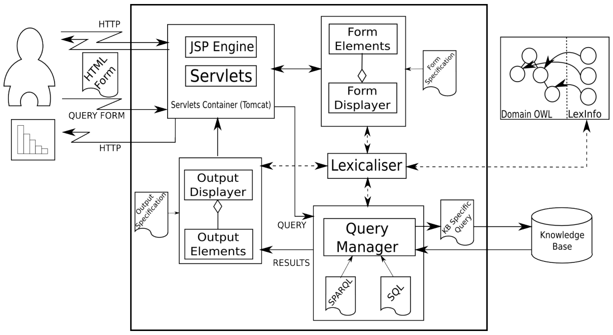


Figure 2: CLOVA general architecture

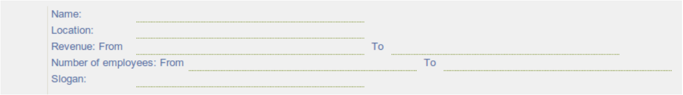
- **Name**: An internal identifier is used to name the input fields for HTML and HTTP requests.
- **Query output**: This defines whether this field will be included in these results. Valid values are *always*, *never*, *ask* (the user could decide wether to include the field in the results or not), *if_empty* (if the field has not been queried it is included in the output), *if_queried* (if the field is queried, it is included in the output) and *ask_default_selected* (the user decides, but as default the field will be shown).
- **Property**: represents the URI for the ontology property to be queried through the field. An indication of `reference=self` in place of a URI means that we are querying the domain of the search. Such queries are useful for querying the lexicalisation of the object being queried or limiting the query to a fixed set of objects.
- **Property Range**: We define a number of types (called *property ranges*) that describe the data that a field can handle. It differs from the data types of RDF or similar in that we also describe how the data should be queried as well. For example, while it is possible to describe both the revenue of a company and the age of an employee as integers in the database, it is not sensible to query revenue as a single value, whereas it is often useful to query age as a single value. These property ranges provide an abstraction of these properties in the data and thus support the generation of appropriate forms

and queries. The following property ranges are built-in into CLOVA:

- *String*, *Numeric*, *Integer*, *Date*: Simple data-type values. Note that *String* is intended for representing language-independent strings, e.g. IDs, not natural language strings. The numeric and date ranges are used to query precise values like "age" and "birth date".
- *Range*, *Segment*, *Set*: These are defined relative to another property range and specify how a user can query the property in question. *Range* specifies that the user should query the data by providing an upper and/or lower bound, e.g. "revenue", "number of employees". *Segment* is similar but requires that the developer divides the data up into pre-defined intervals. *Set* allows the developer to specify a fixed set of queriable values, e.g. "marital status".
- *Lexicalised Element*: Although we assume all data in the source is defined by URIs, it is obviously desirable that the user can query the data using natural language. This property range in fact allows to query for URIs through language-specific strings that need to be resolved by the system to the URI in question. The strings introduced into this field are processed by the lexicaliser to find the URI to which they belong which is then used in the corresponding queries. For example, locations can have different names in different languages, e.g. "*New York*" and "*Nueva York*", but the URI in the data source should be the same.
- *Complex*: A complex property is considered to be a property composed of other sub-properties. For example, searching for a "key person" within a company can be done by searching for properties of the person, e.g., "name", "birth place". This nested form allows us to express queries over the structure of an RDF repository or other data source.
- *Unqueriable*: For some data, methods for efficient querying cannot be provided, especially binary data such as images. Thus we defined this field to allow the result to still be extracted from the data source and included in the results.

The described property ranges are supported natively by CLOVA, but it is also possible to define new property ranges and include them in the SFS XML document. The appropriate implementation for a form display element that can handle the newly defined property range has to be provided of course (see Section 3.2.2).

- **Rendering Properties**: There is often information for a particular rendering that cannot be provided in the description of the property ranges alone. Thus, we allow for a set of context specific properties to be passed to the rendering engine. Examples of these include the use of auto-completion features or an indication of the type of form element to display, i.e. a *Set* can be displayed as a drop-down list, or as a radio button selection.



Name:
Location:
Revenue: From To
Number of employees: From To
Slogan:

Figure 3: HTML form generated for a SFS document

The SFS document is in principle similar to the concept of a "lens" in the Fresnel display vocabulary [2] in that it describes the set of fields in the data that should be used for display and querying. However, by including more information about methods for querying the data, we provide a description that can be used for both presentation and querying of the data.

Example: Suppose that we want to build a small web application that queries an ontology with information about companies stored in an RDF repository. The application should ask for company names, companies' revenue, and company locations. The syntax of a SFS XML document for that application is shown below:

```
<!--xmlns:dbpedia="http://dbpedia.org/ontology/"-->
<form domain="dbpedia:Company">
    <fields>
        <field name="Name" output="ALWAYS">
            <property reference="self"/>
            <property-range>
                <lexicalised-property-range/>
            </property-range>
            <rendering context="html">
                <property name="autocompletion" value="yes"/>
            </rendering>
        </field>

        <field name="Location" output="ASK">
            <property uri="&dbpedia;Organisation/location"/>
            <property-range>
                <lexicalised-property-range/>
            </property-range>
        </field>

        <field name="Revenue"  output="ASK_DEFAULT_SELECTED">
            <property uri="&dbpedia;Organisation/revenue"/>
            <property-range>
                <ranged-property-range>
                   <continuous-property-range>
                       <min>0</min>
                   </continuous-property-range>
                </ranged-proprety-range>
            </property-range>
        </field>
    </fields>
</form>
```

3.2.2 Form Displayer

The form displayer consists of a set of *form display elements* defined for each property range. It processes the SFS by using these elements to render the fields in a given order. The implementation of these elements is dependent on the output method. The form display elements are rendered using Java code to convert the document to XHTML[4].

Figure 3 shows an example of rendering of an SFS which includes the fields in the example above. In this rendering the field "name" is displayed as a text field as it refers to the lexicalisation of this company. The location of a company for instance is represented as a text field. However, in spite of the fact that the data is represented in the data source as a language independent URI, the user can query by specifying

[4] The CLOVA project also provides XSLT files to perform the same task

the name of the resource in their own language (e.g., a German user querying “München” receives the same results as an English user querying “Munich”). Finally, the revenue is asserted as a continuous value which is queried by specifying a range and is thus rendered with two inputs allowing the user to specify the upper and/or lower bounds of their query. A minimum value on this range allows for client-side data consistency checks. In addition, check boxes are appended to fields in order to allow users to decide if the fields will be shown in the results, according to the output parameter in the SFS.

3.2.3 *Query Manager*

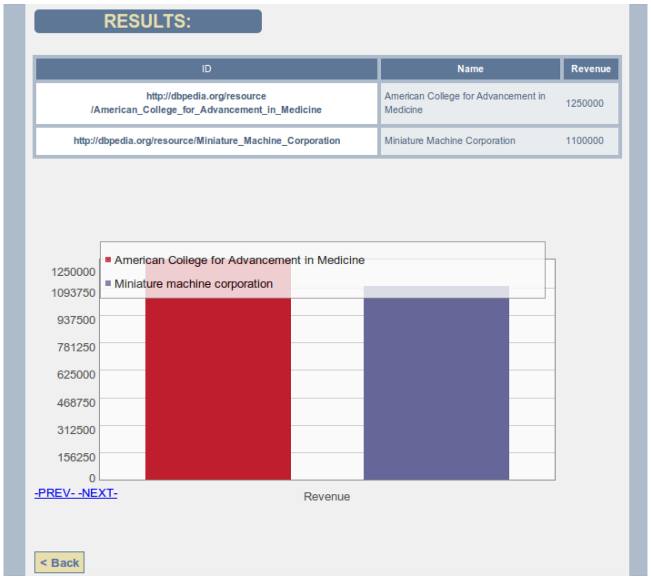
Once the form is presented to the user, he or she can fill the fields and select which properties he or she wishes to visualise in the results. When the query form is sent to the Query Manager, it is translated into a specific query for a particular knowledge base. We have provided modules to support the use of SQL queries using JDBC and SPARQL queries using Sesame [3]. We created an abstract query interface which can be used to specify the information required in a manner that is easy to convert to the appropriate query language allowing us to change the knowledge base, ontology and back end without major problems. The query also needs to be preprocessed using the *lexicaliser* due to the presence of language-specific terms introduced by the user which need to be converted to language independent URIs.

3.2.4 *Output Displayer*

Once the query is evaluated, the results are processed by the *output displayer* and an appropriate rendering shown to the user. The displayer consists of a number of display elements, each of which represents a different visualisation of the data, including not only simple tabular forms, but also graphs and other visual display methods. As with the form displayer, all of these elements are lexicalised in the same manner as the form displayer.

In general we might restrict the types of data that components will display as not every visualisation paradigm is suitable for any kind of data. For example, a bar chart showing foundation year and annual income would be both uninformative and difficult to display due to the scale of values. For this reason we provide an *Output Specification* to define the set of available display elements and sets of values they can display. These output specifications consist of a list of output elements described as follows:

- **ID**: Internal identifier of the output element displayed.
- **URI**: A reference to the output resource specified as a URI.[5]
- **Fields**: The set of fields used by this element. These should correspond by `name` to elements in the SFS.
- **Display properties**: Additional parameters passed to the display element to modify its behaviour. Some of these parameters include the possibility to ignore incomplete data, or to define the subtypes of a chart to display. These parameters are class dependant so that each output element has its own set of valid parameters.



ID	Name	Revenue
http://dbpedia.org/resource /American_College_for_Advancement_in_Medicine	American College for Advancement in Medicine	1250000
http://dbpedia.org/resource/Miniature_Machine_Corporation	Miniature Machine Corporation	1100000

🟥 American College for Advancement in Medicine
🟪 Miniature machine corporation
1250000
1093750
937500
781250
625000
468750
312500
156250
0
-PREV- -NEXT-
Revenue
< Back

Figure 4: HTML result page for the example

The following output specification defines two output elements to show results.

```
<!-- xmlns:clova="jar:file:clova-html.jar!/clova/html/output/"
   xmlns:dbpedia="http://dbpedia.org/ontology/"-->
 <output>
    <elements>
        <element id="HTable" URI="&clova;HTableDisplayElement">
            <fields>
                <all/>
            </fields>
        </element>
        <element id="BarChart" URI="&clova;GraphDisplayElement">
            <fields>
                <field name="revenue"/>
            </fields>
            <display>
                <property name="Type" value="barChart"/>
            </display>
        </element>
    </elements>
</output>
```

The first element displays a table containing all the results returned by the query, while the second output element shows a bar chart for the property “Revenue”. The HTML output generated for a given output specification containing the above mentioned descriptions is shown in Figure 4.

3.2.5 *Lexicaliser*

Simple lexicon models can be provided by language annotations, for example RDF’s `label` and SKOS’s `prefLabel`, and developing a lexicaliser is then as simple as looking up these labels for the given resource URI. This approach may be suitable for some tasks. However, we sometimes require lexicalisation using extra information about the context and would like to provide lexicalisation of more than just URIs, e.g. when lexicalising triples. While RDF labels can be attached to properties and individuals for instance, there is no mechanism that allows to compute a lexicalization for a triple by composing together the labels of the property and the individuals. This is a complex problem and we will leave a full investigation and evaluation of this for future work.

[5] These can reference Java classes by linking to the appropriate class file or location in a JAR file

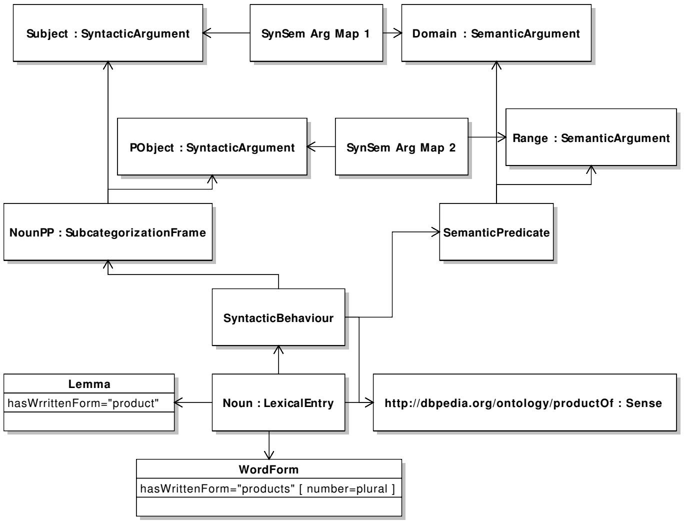

Subject : SyntacticArgument
SynSem Arg Map 1
Domain : SemanticArgument
PObject : SyntacticArgument
SynSem Arg Map 2
Range : SemanticArgument
NounPP : SubcategorizationFrame
SemanticPredicate
SyntacticBehaviour
Lemma
hasWrrittenForm="product"
Noun : LexicalEntry
http://dbpedia.org/ontology/productOf : Sense
WordForm
hasWrittenForm="products" [number=plural]

Figure 5: A simplified example of a LexInfo aggregate

Furthermore, it is often desirable to have fine control over the form of the lexicalisation, for example, the ontology label may be "company location in city". However, we may wish to have this property expressed by the simpler label "location". By using a lexicon ontology model we can specify the lexicalisation in a programmatic way, and hence adapt it to the needs of the particular query interface. For these reasons we primarily support lexicalisation through the use of the LexInfo [4] lexicon ontology model and its associated API[6], which is compatible with the LMF Vocabulary [5].

The LexInfo model:

A LexInfo model is essentially an OWL model describing the lexical layer of an ontology specifying how properties, classes and individuals are expressed in different languages. We refer to the task of producing language-specific representation of elements in the data source including triples as *lexicalisation* of the data. The corresponding LexInfo API organises the lexical layer mainly by defining so called *aggregates* which describe the lexicalisation of a particular URI, specifying in particular the lexico-syntactic behaviour of certain lexical entries as well as their interpretation in terms of properties, classes and individuals defined in the data. An aggregate essentially bundles all the relevant individuals of the LexInfo model needed to describe the lexicalization of a certain URI. This includes a description of syntactic, lexical and morphological characteristics of each lexicon entry in the lexicon. Indeed, each aggregate describes a lexical entry together with its lemma and several word forms (e.g. inflectional forms such as the plural etc.). The syntactic behaviour of a lexical entry is described through subcategorization frames making the required syntactic arguments explicit. The semantic interpretation of the lexical entry with respect to the ontology is captured through a mapping ("syn-sem argument map") from the syntactic arguments to the semantic arguments of a semantic predicate which stands proxy for an ontology element in the ontology. Finally the aggregate is linked through a `hasSense` link to the URI in the data layer it lexicalises. An example of an aggregate is given in figure 5. For details the interested reader is referred to [4].

LILAC:

In order to produce lexicalisations of ontology elements from a LexInfo model we use a simple rule language included with the LexInfo API called LILAC (LexInfo Label Analysis & Construction). A LILAC rule set describes the structure of labels and can be used for both generating the lexicon from labels and generating lexicalisations from the lexicon. In general we assume that lexicons are generated from some set of existing labels, which may be extracted from annotations in the data source, e.g., RDFS's `label`, from the URIs in the ontology or from automatic translations of these labels from another language. The process of generating aggregates from raw labels requires that first the part of speech tags are identified by a tagger such as TreeTagger. Then, the part-of-speech tagged labels are parsed using a LR(1)-based parser (see [1]). The API then handles these parse trees and converts them into LexInfo aggregates.

LILAC rules are implemented in a symmetric manner so that they can be used to both generate the aggregates in the lexicon ontology model (e.g. by analysing the labels of a given ontology) as well as lexicalise those aggregates.

A simple example rule for a label such as "revenue of" is:

```
Noun_NounPP -> <noun> <preposition>
```

This rule states that the lexicalisation of a `Noun_NounPP Aggregate` is given by first using the written form of lemma of the "noun" of the aggregate followed by the lemma of "preposition" of the aggregate. LILAC also supports the insertion of literal terms and choosing the appropriate word form in the following manner:

```
Verb_Transitive -> "is" <verb> [ participle,
    tense=past ] "by"
```

This rule can be used to convert a verb with transitive behaviour into a passive form (e.g., it transforms "eats" into "is eaten by").

LILAC can create lexicalisations recursively for phrase and similar, for example to lexicalise an aggregate for "yellow moon", the following rules are used. Note that in this cases the names provided by the aggregate class are not available so the name of the type is used instead:

```
NounPhrase -> <adjective> <NounPhrase>
NounPhrase -> <noun>
```

The process for lexicalisation proceeds as follows: for each ontology element (identified by a URI) that needs to be lexicalised, the LexInfo API is used to find the lexical entry that refers to the URI in question. Then the appropriate LILAC rules are invoked to provide a lexicalization of the URI in a given language.

As this process requires only the URI of the ontology element, by changing the LexInfo model and providing a reusable set of LILAC rules the language of the interface can be changed to any suitable form. It is important to emphasize that the LILAC rules are language-specific and thus need to be provided for each language supported.

Another issue is that we desire that our users are capable of searching for elements by their lexicalised form. LexInfo can support this as well. This involves querying the lexicon for

[6] Available at `http://lexinfo.googlecode.com/`

all lexical entries that have a word form matching the query and returning the URI that the lexical entry is associated to. Once we have mapped all language-specific strings to URIs, the query can be handled using the query manager as usual. For example if the user queries for "food" then the LexInfo model could be queried for all lexical entries that have either a lemma or word form matching this literal. The URIs referred to by this word can then be used to query the knowledge base. This means that a user can query in their own language and expect the same results, for example the same concept for "food processing" will be returned by an English user querying "food" and a Spanish user querying for "alimento" (part of the compound noun "Procesado de los alimentos").

3.3 CLOVA for company search

We developed a search interface for querying data about companies using CLOVA, which is available at `http://www.sc.cit-ec.uni-bielefeld.de/clova/demo`. For this application we used data drawn from the DBPedia ontology, which we entered into a Sesame store. We used the labels of the URIs to generate the lexicon model for English, and used the translations provided by DBPedia's wikipage links (themselves derived from WikiPedia's "other languages" links), to provide labels in German and Spanish. As properties were not translated in this way, the translations for these elements were manually provided. These translations were converted into a LexInfo model through the use of about 100 LILAC rules. About 20 of these rules were selected to provide lexicalisation for the company search application. In addition, we selected the form properties and output visualisations by producing a semantic form specification as well as an output specification. These were rendered by the default elements of the CLOVA HTML modules, and the appearance was further modified by specifying a CSS style-sheet. In general, the process of adapting CLOVA involves creating a lexicon, which could be a LexInfo model or a simpler representation such as with RDF's `label` property, and then producing the semantic form specification and output specification. Adapting CLOVA to a different output format or data back end, it requires implementing only a set of modest interfaces in Java.

4. CONCLUSION

We have presented an architecture for querying semantic data in multiple languages. We started by providing methods to specify the creation of forms, the querying of the results and presentation of the results in a language-independent manner through the use of URIs and XML specifications. By creating this modular framework we provide an interoperable language-independent description of the data, which could be used in combination with a lexicalisation module to enable multilingual search and querying. We then separated the data source into a language-independent data layer and a language-dependent lexical layer, which allows us to modularise each language and made the lexical information available separately on the semantic web. In this way we achieved all the requirements we set out in Figure 1. We described an implementation of this framework, which was designed to transform abstract specifications of the data into HTML pages available on the web and performed its lexicalisations by the use of LexInfo lexicon ontology models [4] providing fine control on the lexicalisations used in a particular context.

Acknowledgements

This work has been carried out in the context of the Monnet STREP Project funded by the European Commission under FP7, and partially funded by the "Consejería de Innovación Ciencia y Empresa de Andalucía" (Spain) under research project P06-TIC-01433.

5. REFERENCES

[1] A. Aho, R. Sethi, and J. Ullman. Compilers: principles, techniques, and tools. *Reading, MA,*, 1986.

[2] C. Bizer, R. Lee, and E. Pietriga. Fresnel: A browser-independent presentation vocabulary for rdf. In *Proceedings of the Second International Workshop on Interaction Design and the Semantic Web, Galway, Ireland.* Citeseer, 2005.

[3] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. *Lecture Notes in Computer Science*, pages 54–68, 2002.

[4] P. Buitelaar, P. Cimiano, P. Haase, and M. Sintek. Towards linguistically grounded ontologies. In *Proceedings of the European Semantic Web Conference (ESWC)*, pages 111–125, 2009.

[5] G. Francopoulo, N. Bel, M. George, N. Calzolari, M. Monachini, M. Pet, and C. Soria. Lexical markup framework (LMF) for NLP multilingual resources. In *Proceedings of the workshop on multilingual language resources and interoperability*, pages 1–8. Association for Computational Linguistics, 2006.

[6] A. S. M. L. Hoque and M. Arefin. Multilingual data management in database environment. *Malaysian Journal of Computer Science*, 22(1):44–63, 2009.

[7] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou. Ontology visualization methods: a survey. *ACM Computing Surveys (CSUR)*, 39(4):10, 2007.

[8] A. Kumaran and J. R. Haritsa. On database support for multilingual environments. In *Proceedings of the IEEE RIDE Workshop on Multilingual Information Management,*, 2003.

[9] A. Miles, B. Matthews, M. Wilson, and D. Brickley. SKOS Core: Simple knowledge organisation for the web. In *Proceedings of the International Conference on Dublin Core and Metadata Applications*, pages 12–15, 2005.

[10] C. Peters, T. Deselaers, N. Ferro, J. Gonzalo, G. F. Jones, M. Kurimo, T. Mandl, A. Peñas, and V. Petras. *Evaluating Systems for Multilingual and Multimodal Information Access*, volume 5706. Springer, 2008.