PFC

Miguel Parramón
Dept. Llenguatges i Sistemes Informàtics
Universitat Politecnica de Catalunya
parramon@lsi.upc.edu

June 19, 2011
Chapter 1

Abstract

The Semantic Web is the new wave of developments on the World Wide Web pushed forward by millions of developers and heralded by the W3C\(^1\). These new standards aim to provide the web with semantics, so that automated programs can act on data the same way as persons do, together with the open data philosophy, that aims to liberate all kinds of data to the web for the benefit of mankind. On this frameset, we have developed two distinct tools that interact with the Pat.mapa cultural heritage ontology built by the Culture Department of the Generalitat de Catalunya. The Pat.Mapa Extract and Review tool extracts events from texts attached to this semantic database, and lets the end user review them before adding them; while the Pat.mapa Ontology Visualizer lets the end user interact with this semantic database in a intuitive way. We hope that these tools will help everyone on the migration to a semantic web, as we aim to generalise them to work with any semantic database.

\(^1\)http://www.w3.org/
Chapter 2

Introduction

This chapter will describe the context of the project, and will summarise the work that will be explained in detail through it.

The Pat.mapa cultural heritage semantic database of Catalonia was created by the Culture Department of the Generalitat de Catalunya \(^1\), following the example of the Cantabria Heritage Ontology \([6]\), and thus the CIDOC standard \([2]\). It was generated by the union of three Catalan cultural information databases: one of architectural elements, one of archaeological ones, and one of museums.

The Pat.mapa semantic database is structured using the Catalonia geographical hierarchy, detailing the counties, the municipalities that form them, and the elements (architectural, archaeological and museums) that appear in each of them. The information available about these elements is their geographical information via GPS coordinates (and addresses, in case of the museums), their epoch (Prehistory, Modern Age, et.), and their style or type (romantic, Gothic, etc.), among others.

This semantic database is used as the knowledge base of some of the features of the Pat.mapa website\(^2\):

\(^1\)www.gencat.cat/cultura

\(^2\)http://patmapa.gencat.cat/
The other features on this web page: texts, images or videos, come from other non-semantic databases. These texts, written in Catalan, describe each element of the semantic database, with data about the structure of the element, its history, and important events related to it.

One part of this project is the development of the Pat.Mapa Extract and Review tool (PMER) that will be used to extract these events that appear in textual form and add them to the semantic database in a structured way. This will serve to enrich this semantic database with a lot of new relationships between elements, that will be based on the actors and dates that appear on them. This part of the project has been developed by the Computer Languages and Systems (LSIhttp://lsi.upc.edu) department of the Universitat Politecnica de Catalunya³.

The other part of this project will be the development of a graph-based visualization tool for this semantic database, so that a end user can navigate through the elements in an intuitive way and see the relations between them. This tool will be integrated with the Pat.mapa website together with the available map and list view of the elements, as part of a joint project between the i2CAT Foundation ⁴, iSOCO ⁵ and the Culture Department of the Generalitat de Catalunya, having i2CAT the main job assigned in our offices in Citilab⁶.

³http://upc.edu
⁴www.i2cat.net
⁵http://www.isoco.com/
⁶http://www.citilab.eu
Here we present the general structure of this document:

- In the Terminology chapter we introduce some key concepts needed to understand this document.
- In the Requirements chapter we describe the needs of the users of the tools that will be developed, and the ways that we will cover it.
- The Use Cases section show the use cases for the two tools that comprise this project, also detailing each type of user.
- After this, we present an overview of the project, detailing the two tools that form it: The Pat.Mapa Extract and Review tool (PMER), used to extract events from texts and review them in order to add them to the semantic database; and the Pat.mapa Ontology Visualizer (PatViz), a tool to navigate in a visual way through this semantic database.
- Later, in the Implementation section, we describe the tools used to build this project, diving into implementation details if necessary, to provide a deep understanding about interesting developments that have been accomplished.
- Afterwards, in the Related work section, we present some tools developed by other groups that share functional features with this project, or that served as inspiration for it.
- Next is the Workplan section, where the full project’s workplan is detailed, with individual tasks and schedules.
- The Related work section shows our investigation into the state of the art of the fields of work that we will be covering.
- In Future work section, we show the future extension paths for what has been designed in this project, taking into account the experience gathered in its development time, and the new possibilities that these tools permit.
- We then present several files that have been produced through this project’s timeline, in the Appendices section
- As an extra appendix we present a survey of Natural Language Interfaces to Ontologies, which was written at the start of this project as introduction to this field of work.
- Finally, we wrap up the project in the Conclusions section, by reflecting upon this project’s investment, accomplishments, and reach in the community.
Chapter 3

Terminology

The key concepts of this project’s domain of study are described below.

**Natural Language**: A human written or spoken language, as opposed to a computer language. Due to its natural evolution and general use, it lacks the definition and unambiguousness of computer languages.

**Ontology**: A formal representation of the knowledge of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to describe the domain. They are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, and more fields. They are formed by classes, attributes of these classes, relations between classes, and instances of these classes; and they follow a triplet structure: subject-predicate-object (explained below). The more common semantic database languages used to describe them are OWL, RDF and F-Logic. Some known ontologies are Dublin Core, DOLCE, Wordnet, or CIDOC CRM.

**SPARQL**: SPARQL stands for SPARQL Protocol and RDF Query Language, it is an RDF query language to query ontologies in RDF. A query consists of triple patterns, conjunctions, disjunctions, and optional patterns. Example for question: “What are all the country capitals in Africa?”:

**Triplet**: A triple is the basic element of a semantic database, consisting of a subject, a predicate, and an object, such as “Bob is 35” or “Bob knows Fred”.

**CIDOC CRM**: The CIDOC Conceptual Reference Model (CRM) provides an extensible semantic database for concepts and information in cultural heritage and museum documentation. It is the international standard (ISO 21127:2006) for the controlled exchange of cultural heritage information.[1] Archives, libraries, museums, and other cultural institutions are encouraged to use the CIDOC CRM to enhance accessibility to museum-related information and knowledge.

**RDF**: The Resource Description Framework, a metadata data model used to describe ontologies. The predominant query language for RDF graphs is SPARQL.

**WordNet**: Lexical database for the English language, which groups English words into sets of synonyms called synsets, to support automatic text analysis and artificial intelligence applications. As of 2006, the database contains 155,287 words organized in 117,659 synsets for a total of 206,941 word-sense pairs.
Chapter 4

Requirements

4.1 PMER

There are several requirements for this tool. We explain them below:

4.1.1 Recall over automatization

Due to the PMER tool projected user base, which is documentarists and sectors that value precision and accuracy over speed and automatization, this tool cannot be programmed to the point where a probability of error will be added to the decisions taken by the extractor. Therefore, all the decisions that the extractor tool makes are certain to only discard possible events that are really not relevant to the application, like the ones that only have a transitive verb. The other decisions (having dates instead of just numbers, having person name entities instead of just name entities), give more priority to the possible event, but are never used to discard possible events. This can be summarized by saying that this tool aims to have maximum recall, at the expense of precision and automatization.

4.1.2 Automatization of repetitive tasks

The PMER tool is meant to deprive the user base (most of them experts in their area) of any kind of repetitive tasks that make time invested in the task less efficient. That means that this tool will be automatized at the maximum (without losing recall) by minimizing the number of interactions per possible event review, and minimizing the time of the interactions that take place. This will be done by adding an array of keyboard shortcuts, precaching every path taken to minimize time between client-server calls, and joining several operations in one interaction (choosing several actors or dates at the same time by following the syntactic tree structure).
4.1.3 Gentle learning curve

The end users of this tool may have limited experience with computers. Therefore, the tool will have an interactive and static (printed) tutorial, and will be designed in a user friendly way that will also help locate errors and correct them. This requirement will be fulfilled in following iterations of the tool.

4.2 PatViz

4.2.1 Ease of use

The main requirement of the PatViz tool is that, since it is aimed for users that are just looking for information on the Internet, it has to be as easy to use as possible. We will do this by minimizing the kinds of interaction (only one kind of click interaction, and textual input), using the standards of current web design, look and feel, and gently introducing the user to the tool’s features with tooltips (this will be added in subsequent iterations of the project).

4.2.2 Speed

This tool has to be built for speed, since in the current fast paced environment that the Internet is, something that takes more than one second to load will make you lose a majority of potential users. We will do this by using nimble and streaming tools and protocols, and lazily processing most of the data before showing it to the user.

4.2.3 Extensibility

The PatViz tool has to be extensible so that everyone can profit from this development, not only the field of cultural heritage of Catalunya. We will do this by providing an interface and API that are completely independent of this specific domain, and that only need to be configured through external files on execution time to adapt them to new domains.
Chapter 5

Use cases

Here we present the use cases defined for the PatViz semantic database visualization tool, and the PMER extract and review tool.
5.1 PatViz

PatViz Use Cases

Diagram showing navigation and use cases for End User and API User.
5.1.1 End User

The end user is defined as the one that will user the tool to navigate visually through the semantic information via its graph representation.

- **Navigation**
  The end user will be able to navigate through the semantic database via its graph representation. This covers clicking on any node to get the related nodes to it, and seeing the relationship between the nodes currently in screen.

- **Zoom/Pan**
  The end user will be able to zoom and pan to modify the viewport of the tool. The nodes not currently in screen will be visible when the user puts them again in the viewport.

- **Textual Search**
  The end user will be able to search the nodes by their names via a text box. This functionality will not be implemented due to time constraints.

- **Filtering/clustering**
  The end user will be able to filter and cluster the nodes according to their type or relationship.

- **More Info**
  The end user will be able to get more information about the concept that a node represents: text, images, videos, etc.

5.1.2 API User

The API user is defined as the one that will use the API provided by this tool to access the same information that can be seen in a human-friendly way through the visualization tool.

- **Home graph**
  The API user will be able to consult the home graph, with Catalunya and its counties.

- **Related Concepts**
  The API user will be able to consult, given a node id, its related concepts.

- **Textual Search**
  The API user will be able to consult, given a string of text, the nodes which name contains this string. This functionality will not be implemented due to time constraints.

- **Filtering**
  The API user will be able to, given a node id, filter its related concepts via a string of text, which will be compared with the appropriate attributes of the node (type, architectonic style, etc).

- **Info**
The API user will be able to, given a node id, get more information about it (text, images, videos, etc).

5.2 PMER

PMER Use Cases

5.2.1 End User

The end user is defined as the one that will use the tool to extract possible events from texts, review them and add them to the semantic database.

- Extract possible events from a text
  
  The end user will be able to, given a URI to a text document, extract its possible events.

- Verb-type mapping
The end user will be able to, given a verb that forms part of at least one possible event, map it to an event type. This will map in cascade all the verbs with the same level in the semantic hierarchy to this event type.

• Review a possible event

The end user will be able to review a possible event, choosing the correct actor(s), date(s), event type and text.

• Review and normalize verbs/dates

The end user will be able to normalize verbs and dates, to avoid and similar actors and dates that refer to the same concept.

• Insert a reviewed event to the semantic DB

The end user will be able to insert a reviewed event to the semantic DB.
Chapter 6

Overview of the project

Here we dig deeper into the two tools developed for this project, and their context.

6.1 Pat.mapa Extract and Review tool: PMER

The PMER tool can be thought of two different modules: the extract module, that will gather the possible events that appear on the aforementioned texts; and the review module, which enables the user to specify the events that will be used to enrich the semantic database. The features of each of these modules are explained below.

6.1.1 Extract module

The Pat.mapa website offers, among other things, textual information about the cultural heritage elements of Catalunya. One example of these texts can be seen below:
Given one of these texts, this tool extracts the possible events that it recognizes following a set of heuristics. One possible event is composed of:

- The analyzed phrase
- The verbs of the sentence, each one with its EuroWordNet Top Concept Ontology (TCO) properties
- The possible actors who appear in the sentence
- The possible dates that appear in the sentence
- A priority value, calculated according to the extracted components

The extraction of possible events is done in the following way:

1. Separation of the texts in phrases
2. Morphological category tagging
3. Name Entities recognition and classification
4. Word sense disambiguation
5. Syntactic parsing of the phrase
6. Component extraction, going through the syntactic tree and gathering them using the following criteria:
   - Actors: the person name entities are gathered, same as with other name entities if their length is more than 1, to discard false positives
Verbs: the main verbs are selected, and the semi-auxiliary ones if they are head of
their syntagm. For each verb, we collect its properties from the TCO.

Dates: the dates recognized by the NLP library are collected, and also the recognized
numbers if they are head of syntagm.

7. Construction of the selection tree with the lengths of each word and syntagm, for further
component selection in the reviewer module.

8. Calculation of the priority of the possible events, giving more points if person name entities
or dates are found, and discarding phrases without components

Example of the extraction of a possible event from a phrase of the text:

La diferencia estilística es fa palesa a la façana, aixecada entre el
1650 i el 1757, segons projecte de Martí Abaria simplificat per
Antoni Ferrer, dintre dels esquemes barrocs.

1. Morphological category tagging and Name Entities recognition: see appendix examples/morpho.pdf
2. Word sense disambiguation: we choose the most probable sense for each word.
3. Syntactic parsing of the phrase: see appendix examples/tree.svg
4. Component extraction, going through the syntactic tree and gathering them using the
following criteria:
   • Actors: the person name entities are gathered, same as with other name entities if
     their length is more than 1, to discard false positives:
     Martí Abaria
     Antoni Ferrer
   • Verbs: the main verbs are selected, and the semi-auxiliary ones if they are head of
     their syntagm. For each verb, we collect its properties from the TCO:
     aixecada: BoundedEvent:Cause:Existence:Physical,
     simplificat : Cause:Dynamic
   • Dates: the dates recognized by the NLP library are collected, and also the recognized
     numbers if they are head of syntagm.
     1650
     1757

5. Construction of the selection tree with the lengths of each word and syntagm, for further
component selection in the reviewer module.

6. Calculation of the priority of the possible events, giving more points if person name entities
or dates are found, and discarding phrases without components:

   Initial priority: -1
   Actors: person name entity(es) appear -> +1
   Verbs: at least one non-auxiliary verb -> +1
   Dates: Numbers found, no dates found: -> +0
   Total priority: 1

Possible event found! summary:
Of these extracted possible events, the actors and dates are saved as singular objects, meaning that, if there are two possible events that talk about the same actor or date, they will point to the same actor or date object. This is done so that the end user can later review the possible events by actors or dates, and also review these objects, join similar ones, and so forth.

The verbs are also treated in a specific way to help further review. Each verb is saved as a singular object (multiple possible events with the same verb point to the same object), and each verb is also saved with its TCO type. This is done so that when the end user maps each verb to an event type, he is really mapping the corresponding TCO type to an event type. This is done in order to minimize the user's work in the verb to event type mapping. Here you can see a simplified diagram of the verb : TCO type : event type:

6.1.2 Review module

This tool is used to review the possible events extracted from the texts, creating events to add to the semantic database. An event has the following structure:

Type: Creation / Modification / Parts Extension / Parts Removal / Destruction
Related to: Element x
Related to: [Actor(s)]
It happens at [Date(s)]
Has note: [textual explanation of the details]

The review process can be done going through the possible events by relation to a text or an actor. This review can be done in the following ways:

- Starting from the possible events related to a text
- Starting from the possible events related to an actor

The general procedure to follow is:
1. Verb linking: The user relates each type of verb of the verbs of the possible events to review with the events types of the semantic database. For example:

- Verb: construir
- TCO Type: BoundedEvent : Cause : Existence : Physical
- Assigned event type: E63 Beginning of existence

Thus, all verbs with the TCO type "BoundedEvent: Cause: Existence: Physical" will be related to the event type "E63 Beginning of Existence."

Due to the relationship of the verb types with the verbs, it is not necessary to review all of them, since as we assigned event types to each verb type, we are reviewing several verbs at the same time. We can also remove verb types if they are not relevant, and this remove related verbs and possible events in cascade, if necessary.

2. Review of possible events: This process can be done consecutively, going through the possible events related to a text or an actor, or one by one, select the possible event you want to review from the the list of available ones.

The process for reviewing a possible event is the following:

- Selection of the text for the note with the selection tree
- Type selection
- Actor(s) selection
- Date(s) selection

When reviewing, you can add more than one event per possible event, or you can ignore the possible event.

Once we have reviewed the possible events and created the events, we can edit, delete, or confirm them for the insertion to the semantic database.

The insertion to the semantic database follows the predefined mapping. In this case, the Event $\Rightarrow$ CIDOC CRM event mapping is the following one:

6.2 Patmapa Ontology Visualizer: PatViz

The PatViz tool is a graph-based visualization tool for the Pat.mapa semantic database, which will enable end users to navigate through the elements in an intuitive way and see the relations between them.
This tool has two parts: the client-side module, and the server-side module. The server-side module (which enables data access) and the communication schema, are the parts developed by the LSI department of UPC, and the ones that we will cover in detail on this document. On the other hand, the client-side module is being developed by i2CAT, and we will describe this part in broad terms.

6.2.1 Interface module

The PatViz user interface is a client-side tool which shows the Pat.mapa semantic database in a visual way by presenting it as a graph. This tool has the following features:

- Click-based navigation
- Physics engine for automatic spatial distribution of nodes
- Color-coded node types
- Automatic and manual zoom
- One node per concept: vertexes with the same source or target nodes are linked correctly.

The up to date stable version can be seen here:
http://84.88.41.29/

And here is the class diagram of this module:
6.2.2 Back-end module: rdfgraph API

The server-side back-end of the PatViz tool is rdfgraph, an API that converts custom SPARQL consults to a semantic database to graph format and enables access to this information in a RESTful way. Currently this API implements the GET action to get information about a node by its ID. It is configured by writing SPARQL queries to get the following information:

- Name of a node
- Type of a node
- For each type T, Nodes related to a node of type T

Here it is presented the SPARQL query for retrieving the related nodes of a node of type "Architectonic Element": its father (a municipality), its spatial coordinates, and its architectonic style:

```sparql
CONSTRUCT {
  pc:%1$s pc:forms_part_of ?father;
  pc:PP24F.coordsX ?cord_x;
  pc:PP25F.coordsY ?cord_y;
  pc:has_style ?style.
}
WHERE {
  {
    pc:%1$s pc:P88B.forms_part_of ?father .
  }
  UNION {
    pc:%1$s pc:PP24F.coordsX ?cord_x;
  }
  UNION {
    ?site pc:P53F.has_former_or_current_location pc:%1$s;
    pc:P2F.has_type ?style .
  }
}
```

The work flow of this module is, given a node N to consult:

1. Fetching of N's type using the "Type" query.
2. Fetching of the statements defining the nodes related to N and their labels, using the query for N's type
3. Conversion of these statements to graph format. For each statement source, label, target:
   (a) Source and Target processing:
      i. Fetching of its name using the "Name" query
      ii. Fetching of its type using the "Type" query
   (b) New vertex source, label, target added to the graph
4. Encoding to JSON format and response
6.2.3 Interface - Back-end Communication

The communication between the interface and the back-end of the visualization tool is done via Representational State Transfer (REST) \(^1\), and using a simple adhoc graph format, which has the following structure:

Graph: Array of Vertex
Vertex: \{label, Source, Target\}
Source, Target: Element
Element: \{id, name, type\}
label, id, name, type: primitive types

This format (together with REST) is all that is needed for the Interface module and the Back-end module to talk between them, since there is no state to be shared between them. Examples of these exchanges can be seen in the Implementation section.

\(^1\)http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
Chapter 7

Implementation

Here we will show some interesting implementation details about the tools covered by this project.

7.1 PMER

The PMER tool is implemented as a web application for the end user, using the Ruby Programming Language\(^1\). It uses the web application framework Ruby On Rails\(^2\), which follows the MVC pattern, and the Haml\(^3\) and jQuery\(^4\) libraries for the view templates and part of the user interface, respectively.

7.1.1 Ruby on Rails

Ruby on Rails is an open source web application framework for the Ruby programming language. It uses the Model-View-Controller (MVC) architecture pattern, and includes tools that make common development tasks easier, such as scaffolding that can automatically construct some of the models and views needed for a basic website. It also includes WEBrick, a simple Ruby web server, and Rake, a build system. It offers both HTML and XML as output formats, the latter for RESTful web services. Its philosophy is to emphasize Convention over Configuration (CoC), and the rapid development principle of Don’t Repeat Yourself (DRY).

RoR is separated into various packages: ActiveRecord (an object-relational mapping system for database access), ActiveResource (provides web services), ActionPack, ActiveSupport and ActionMailer. Apart from standard packages, developers can make plugins to extend existing packages. RoR is often installed using RubyGems, a package manager included with current versions of Ruby. It is typically deployed with a database server such as MySQL or PostgreSQL, and a web server such as Apache running the Phusion Passenger module.

\(^1\)http://www.ruby-lang.org/

\(^2\)http://rubyonrails.org/

\(^3\)http://haml-lang.com/

\(^4\)http://jquery.com/
7.1.2 FreeLing

FreeLing\(^5\) an open source language analysis tool suite, released under the GNU General Public License and developed by the TALP Research Center\(^6\) of the UPC\(^7\). These are the FreeLing features used by the PMER tool:

- Text tokenization
- Sentence splitting
- Morphological analysis
- Flexible multiword recognition
- Probabilistic prediction of unknown word categories
- Named entity detection
- Recognition of dates, numbers, ratios, currency, and physical magnitudes (speed, weight, temperature, density, etc.)
- PoS tagging
- Chart-based shallow parsing
- Named entity classification
- WordNet based sense annotation and disambiguation
- Rule-based dependency parsing

7.1.3 FreeLing API for Ruby

For the Natural Language Processing part of the tool, the 3.0 development version of FreeLing library is used. As part of this project, the FreeLing Application Programming Interface (API) for Ruby was implemented, following the implementation of the Python and Perl APIs.

The Ruby API follows the same bindings of the Python API, since these two languages are similar syntax-wise. However, SWIG\(^8\), the tool used to generate these APIs in FreeLing, lacks support for C wide string when using Ruby or Perl as the target language. Since this problem was already solved in the Perl interface specification via a custom template, we followed their path and did the same for Ruby.

This Ruby API is intended to be offered to Freeling’s upstream as part of this project’s deployment.

7.1.4 Haml markup language

Haml\(^9\) Haml (HTML Abstraction Markup Language) is a lightweight markup language that is used to describe the XHTML of any web document without the use of traditional inline coding.

\(^5\)http://nlp.lsi.upc.edu/freeling/
\(^6\)http://www.talp.cat/talp/
\(^7\)http://upc.edu/
\(^8\)http://www.swig.org/
\(^9\)http://haml-lang.com/
It’s designed to address many of the flaws in traditional templating engines, as well as making markup as elegant as it can be. Haml functions as a replacement for inline page templating systems such as PHP, RHTML, and ASP. However, Haml avoids the need for explicitly coding XHTML into the template, because it is itself a description of the XHTML, with some code to generate dynamic content.

This project uses Haml as the templating language for the HTML in the PMER tool, due to its simplicity and cleanliness, along with its very smooth learning curve. We also found a that the barrier of entry for this tool is low, since the only thing that was needed to migrate for ERB, the default Ruby On Rails templating language, was to install a gem to add support for Haml, and run some commands to translate all the existing ERB templates into Haml templates.

Here is an example of one template in ERB and Haml:

**ERB**

```html
<div id="profile">
  <div class="left column">
    <div id="date"><%= print_date %></div>
    <div id="address"><%= current_user.address %></div>
  </div>
  <div class="right column">
    <div id="email"><%= current_user.email %></div>
    <div id="bio"><%= current_user.bio %></div>
  </div>
</div>
```

**Haml**

```haml
#profile
  .left.column
    #date= print_date
    #address= current_user.address
  .right.column
    #email= current_user.email
    #bio= current_user.bio
```

**7.1.5 jQuery**

jQuery is a cross-browser JavaScript library designed to simplify the client-side scripting of HTML. It is the most popular JavaScript library in use today. jQuery’s syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plugins on top of the JavaScript library. Using these facilities, developers are able to create abstractions for low-level interaction and animation, advanced effects and high-level, theme-able widgets. This contributes to the creation of powerful and dynamic web pages.

jQuery is used in the the review module of the PMER tool to make it more intuitive and simple to use. Some of the enhancements are the following:

- Easy selection of the words for the "note" part of the event via the sentence tree

---

10http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/
• Highlighting of the selected concepts on the form in the text
• Keyboard shortcuts for all selections and actions

7.2 PatViz

7.2.1 HTML5

The Interface module of PatViz is implemented using a set of HTML5 tools. HTML5 is the latest revision of the HTML standard (originally created in 1990) and currently remains under development. Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers etc.). It brings the capability of building Rich Internet Applications for modern web browsers without using external plugins (like it was before with Adobe Flash, Java Applets, Microsoft Silverlight, etc).

The data layer of this tool is handled via jQuery, using jStorage for the cache and its JSON capabilities for communication. The internal representation is handled via a hash table of the nodes, so that nodes are easily located and never repeated.

The visual layer is done with Processing.js\(^1\), the JavaScript port of Processing\(^2\), an open source programming language and integrated development environment (IDE) built for the electronic arts and visual design communities with the purpose of teaching the basics of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks. It also uses the traer.physics\(^3\) Processing library for the particle system, springs and attraction.

7.2.2 Java Servlets and libraries

The Back-end module of PatViz is implemented as a set of Java Servlets, using the following libraries:

• Sesame\(^4\), an open-source framework for querying and analyzing RDF data, used as a semantic database repository to consult the Pat.mapa semantic database through SPARQL queries.

• JGraphT\(^5\), a free Java graph library that provides mathematical graph-theory objects and algorithms, used for the graphs’ internal representation.

• google-gson\(^6\), a Java library that can be used to convert Java Objects into their JSON representation, used for encoding the graphs to JSON.

---

\(^1\)http://processingjs.org/
\(^2\)http://www.processing.org/
\(^3\)http://murderandcreate.com/physics/
\(^4\)http://www.openrdf.org/
\(^5\)http://www.jgrapht.org/
\(^6\)http://code.google.com/p/google-gson/
7.2.3 JSON

The communication between the interface and the back-end of the visualization tool is implemented using REST and the adhoc graph format explained in subsection ??.

We present here an example of the requests and responses generated when a user navigates through the semantic database using PatViz:

1. PatViz initialization, render the home graph:
   - Request: GET http://server/rdfgraph/home
   - Response: JSON graph with "Catalunya" node and its counties
   - Result: The home graph is rendered, Catalunya and its counties

![Image of PatViz interface](image.png)

2. Click on the "Alt Camp" County to inspect its relationships:
   - Request: GET http://server/rdfgraph/rel?id=["Alt Camp"’s id]
   - Response: JSON graph with the "Alt Camp" node and its municipalities
   - Result: The new graph is added to the screen, now the municipalities of Alt Camp can be seen connected to it.
3. Click on the "Querol" Municipality to inspect its relationships:
   - Request: GET http://server/rdfgraph/rel?id="Querol”’s id"
   - Response: JSON graph with the "Querol" node and its cultural heritage elements
   - Result: The new graph is added to the screen, now the cultural heritage elements of Querol can be seen connected to it.

   - For each element:
     - Request: GET http://server/rdfgraph/rel?id=[element id]
     - Response: JSON graph with the [element] node and its attributes
     - Result: The element’s attributes are added to the screen, connected to it and the other elements if they are shared.
Here is also a simple example of a graph showing Barcelona, the Sagrada Familia, and Antoni Gaudí, encoded using the JSON graph format:

```json
{
    "graph": [
        {
            "label": "forms part of",
            "source": {
                "id": "1",
                "name": "Sagrada Familia",
                "type": "Architectural Element"
            },
            "target": {
                "id": "2",
                "name": "Barcelona",
                "type": "Municipality"
            }
        },
        {
            "label": "created by",
            "source": {
                "id": "1",
                "name": "Sagrada Familia",
                "type": "Architectural Element"
            },
            "target": {
```
"id": "3",
"name": "Antoni Gaudi",
"type": "Actor"
}
Chapter 8

Workplan

Here you can see the workplan for the complete project:
Chapter 9

Related work

9.1 Visualizer

Expertum\textsuperscript{1} is a visualization tool for complex data, developed by the LSI department of the UPC, which served as aesthetic inspiration for our project.

The Linked Open Data VIZualizator project: LODVIZ\textsuperscript{2} is a web tool to visualize semantic databases, using HTML5 technologies. It is in active development, by Francisco Cifuentes Silva of the Web Semántica Oviedo (WESO\textsuperscript{3}) group. This tool is very similar to PatViz, however it can be used with any RDF repository, but it does not allow navigation between the nodes.

The Visual Data Web group\textsuperscript{4} has released several visual tools for interacting with linked data:

- Relfinder: Extracts and visualizes relationships between objects in RDF data and makes these relationships interactively explorable.
- SemLens: Visual interface that combines scatter plots and semantic lenses. The scatter plots offers a global visualization that can be locally explored with the semantic lenses.
- gFacet: gFacet supports faceted exploration of RDF data by representing facets as nodes in a graph visualization, which can be interactively added and removed in order to produce individual search interfaces.
- fFacet: Hierarchical faceted exploration of RDF data, using known interaction concepts to allow hierarchical faceted exploration of RDF data.

IsaViz \textsuperscript{11} is a visual environment for browsing and authoring RDF models represented as graphs, with the following features:

- 2.5D user interface allowing smooth zooming and navigation in the graph
- creation and editing of graphs by drawing ellipses, boxes and arcs
- RDF/XML, Notation 3 and N-Triple import

\textsuperscript{1}http://laboranovalsi.upc.edu/xpertum/
\textsuperscript{2}http://156.35.98.54/lodviz/index.jsp
\textsuperscript{3}http://www.weso.es/
\textsuperscript{4}http://www.visualdataweb.org/

32
• RDF/XML, Notation 3 and N-Triple export, but also SVG and PNG export

9.2 Extract-Review module

There are several surveys covering the topic of automatic enrichment and construction of ontologies via text [15] [3] [1] [10] [9]. However, we didn’t find information about extraction of an specific type of semantic construction (events) to fit an existint semantic database structure (CIDOC CRM) with a high level of recall. Therefore, and due to time constraints, our tool has been mostly developed using NLP techniques as an adhoc solution for this project.
Chapter 10

Future work

Here are some future extension paths we have found to be interesting for this project’s tools:

10.1 PMER ⇒ RETO

We will further develop the PMER tool to generalize so that it can work with any semantic database. This will be done in two steps: first, a configuration section will be added to choose and configure the semantic database that will be used manually; afterwards, the configuration will be automatized as far as it is possible using AI and NLP tools. This new tool could be referred to as a Text to Ontology Extractor and Reviewer (RETO).

The Review module of this tool can also be enhanced by adding AJAX-style interactions, User management, interface personalization, and more.

10.2 rdfgraph API

The API to consult SPARQL endpoint via converting the answer Triples to the JSON graph format will be enhanced to work with multiple semantic database repositories. The remaining configuration parameters needed to use it will be moved to configuration files so that there will not be a need to recompile the tool to change the endpoint or the API endpoints and their data structure. This will be done by converting the API to a set of LarKC\(^1\) plugins, so that future versions of the tool will depend on this framework for data access and extensibility.

This API will also be extended to be able to modify the target semantic database via the POST and DELETE REST verbs. This will open the door to a very broad range of new application, that would target this API to modify and extend semantic databases.

\(^{1}\)www.larkc.eu
10.3 Extract-review API

The Extract review API in its current state is mostly independent of the semantic database to be enriched. This will be further enhanced to be able to choose the output format (Event - OntoEvent mapping). The ability to extract and review more kinds of information apart from events (spatial descriptions, conceptual explanations) could also be added in future versions.
Chapter 11

Appendixes

In this chapter and attached to this document we present several more documents about the development of this project:

- Several documents about the state of the art of the complex data visualization
- Documentation about the CIDOC CRM and FRBR\(^1\) standards
- Joint documentation with the Department of Culture of the Generalitat de Catalunya
- Documentation about use of project management tools.
- Information about internal developments about remapping the Pat.mapa ontology to better fit the CIDOC CRM standard.

11.1 "Documentation of the project" folder

In this folder we present the documentation that we have gathered through this project’s development time. This includes:

11.2 "PatViz" folder

We present the complete code base of the PatViz client-side tool. This shows the evolution from a full-fledged Java applet, to an Adobe Flash client-side tool working with the rdfgraph, to an HTML5 client-side tool.

11.3 "PMER" folder

In this folder you can find all the code for the PMER tool, showing the evolution from two different modules: the extract module using Java and the review module written on Ruby, to

\(^1\)http://www.ifla.org/en/publications/functional-requirements-for-bibliographic-records
the unified Ruby version currently in development.

11.4 "rdgraph" folder

This folder contains the full code base of the rdfgraph API, in its current state.

11.5 Original Workplan and Chronogram of the project

These documents entail to the original time previsions of this project; we show it to give a perspective by comparing the first-day ideas to where we have gotten to.
ANNEX I.

CONTRACTE DE PRESTACIÓ DE SERVEIS D’INVESTIGACIÓ TECNOLÒGICA PER AL DESENVOLUPAMENT DEL PROJECTE DE RECERCA I INVESTIGACIÓ PAT.MAPA 2010
Pla de Treball

PT1 Coordinació

<table>
<thead>
<tr>
<th>Numero de paquet de treball</th>
<th>1</th>
<th>Data Inici/Final</th>
<th>M1-M10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Títol del paquet de treball</td>
<td>Coordinació</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Objectius

- Coordinar les activitats del projecte;
- Assegurar la qualitat i la puntualitat dels documents a entregar;
- Establir les metodologies de treball i comunicació;
- Estimular la realització de les tasques dels PTs;
- Monitorejar i facilitar el progrés continu;
- Identificar i evitar els obstacles que puguin afectar al projecte;
- Avaluat i reportar el progrés del projecte basant-se en ens indicators establerts;

Descripció del treball

Tasca 1.1 Reunió inicial. Aquesta reunió inicia el projecte. Es presenta el pla d'execució de cada PT i es deixa clara la responsabilitat dels membres. Es decideix sobre la metodologia de treball i les eines de comunicació.

Tasca 1.2 Reunions plenàries. Cada mes, utilitzant sistemes de videoconferència o presencials els participants es reuneixen per revisar les activitats en marxa i comunicar les possibles desviacions/ activitats paral·leles / informació relacionada.

Tasca 1.3 Control de progrés. Tasca continua durant tot el projecte orientada a controlar el progrés de les activitats, escriure els informes de progrés i resoldre problemes puntuals que puguin aparèixer.

Tasca 1.5 Qualitat. Revisió de la qualitat de cada document relacionat amb el projecte.

Entregables

<table>
<thead>
<tr>
<th>Id</th>
<th>Títol Entregable</th>
<th>Responsable</th>
<th>Dedicació</th>
<th>Tipus</th>
<th>Nivell de disseminació</th>
<th>Data d'entrega</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1.1</td>
<td>Pla de treball detallat</td>
<td>i2CAT</td>
<td>Document</td>
<td>Privat</td>
<td>M1</td>
<td></td>
</tr>
<tr>
<td>D1.2</td>
<td>Presentació del projecte</td>
<td>i2CAT</td>
<td>Presentació</td>
<td>Públic</td>
<td>M2</td>
<td></td>
</tr>
<tr>
<td>D1.3</td>
<td>Report final</td>
<td>i2CAT</td>
<td>Informe</td>
<td>Privat</td>
<td>M10</td>
<td></td>
</tr>
</tbody>
</table>

Comentaris
PT2 Anàlisi i especificació

<table>
<thead>
<tr>
<th>Numero de paquet de treball</th>
<th>2</th>
<th>Data Inici/Final</th>
<th>M1-M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Títol del paquet de treball</td>
<td>Requeriments d’usuari</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Objectius

- Recollir els requeriments d’usuari;
- Analitzar els requeriments d’usuari;
- Especificar formalment les funcionalitats del sistema.

Descripció del treball

**Tasca 2.1 Requeriments d’usuari.** Recopilar en dos reunions les necessitats del client final;

**Tasca 2.2 Anàlisi de requeriments.** Analitzar els requeriments d’usuari i validar-los amb el mateix usuari.

**Tasca 2.3 Especificació.** Especificar formalment les funcionalitats del sistema, els rols d’usuari i la interacció entre usuari i sistema.

Aquest PT es realitzarà iterant dos cops sobre les tres tasques de cara a deixar clares totes les funcionalitats que ha de tenir el sistema.

### Entregables

<table>
<thead>
<tr>
<th>Id</th>
<th>Títol Entregable</th>
<th>Responsable</th>
<th>Dedicació</th>
<th>Tipus</th>
<th>Nivell de disseminació</th>
<th>Data d’entrega</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2.1</td>
<td>Document de requeriments</td>
<td>I2CAT</td>
<td>Informe</td>
<td>Privat</td>
<td>M1</td>
<td></td>
</tr>
<tr>
<td>D2.2</td>
<td>Especificacions</td>
<td>I2CAT</td>
<td>Document tècnic</td>
<td>Privat</td>
<td>M1</td>
<td></td>
</tr>
<tr>
<td>D2.3</td>
<td>Document PFC1</td>
<td>UPC</td>
<td>Informe</td>
<td>Privat</td>
<td>M2</td>
<td></td>
</tr>
<tr>
<td>D2.4</td>
<td>Document PFC2</td>
<td>UPC</td>
<td>Informe</td>
<td>Privat</td>
<td>M2</td>
<td></td>
</tr>
</tbody>
</table>

Comentaris
### PT3 Disseny i arquitectura

<table>
<thead>
<tr>
<th>Numero de paquet de treball</th>
<th>3</th>
<th>Data Inici/Final</th>
<th>M2 o M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Títol del paquet de treball</td>
<td>Disseny i arquitectura</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Objectius
- Dissenyar la interfície d'úsuaris.
- Dissenyar l'arquitectura del sistema.

#### Descripció del treball

**Tasca 3.1 Disseny interfície.** Donades les funcionalitats detallades al document de especificació, dissenyar la interfície gràfica d'úsuaris.

**Tasca 3.2 Disseny i arquitectura del sistema.** Traslladar les especificacions del sistema a una arquitectura software que les implementi, dividir la implementació en mòduls amb tasques i subtasques.

#### Entregables

<table>
<thead>
<tr>
<th>Id</th>
<th>Títol Entregable</th>
<th>Responsable</th>
<th>Dedicació</th>
<th>Tipus</th>
<th>Nivell de disseminació</th>
<th>Data d'entrega</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3.1</td>
<td>Disseny d'interfície</td>
<td>I2CAT</td>
<td>Vista</td>
<td>Privat</td>
<td>M2</td>
<td></td>
</tr>
<tr>
<td>D3.2</td>
<td>Arquitectura del sistema</td>
<td>I2CAT</td>
<td>Document tècnic</td>
<td>Privat</td>
<td>M3</td>
<td></td>
</tr>
<tr>
<td>D3.3</td>
<td>Arquitectura PFC1</td>
<td>UPC</td>
<td>Document tècnic</td>
<td>Privat</td>
<td>M3</td>
<td></td>
</tr>
<tr>
<td>D3.4</td>
<td>Arquitectura PFC2</td>
<td>UPC</td>
<td>Document tècnic</td>
<td>Privat</td>
<td>M3</td>
<td></td>
</tr>
</tbody>
</table>

#### Comentaris
PT4 Implementació i validació

<table>
<thead>
<tr>
<th>Numero de paquet de treball</th>
<th>4</th>
<th>Data Inici/Final</th>
<th>M3 i M10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Títol del paquet de treball</td>
<td>Producció</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Objectius
- Implementar l’arquitectura del sistema dissenyada en el PT3

Descripció del treball

**Tasca 4.1 Divisió de la implementació en fases.** A priori es preveu dividir la implementació en prototip, producte, producte millorat. En aquesta tasca es decidiran quines funcionalitats / mòduls/ tasques o subtasques s’implementen en cada fase.

**Tasca 4.2 Implementació.** Implementació en 3 fases del sistema.

**Tasca 4.3 Validació.** Validació intermèdia i final del producte.

### Entregables

<table>
<thead>
<tr>
<th>Id</th>
<th>Títol Entregable</th>
<th>Responsable</th>
<th>Dedicació</th>
<th>Tipus</th>
<th>Nivell de disseminació</th>
<th>Data d’entrega</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4.1</td>
<td>Implementació prototip i validació</td>
<td>I2CAT</td>
<td>Codi i informe</td>
<td>Privat</td>
<td>M4</td>
<td></td>
</tr>
<tr>
<td>D4.2</td>
<td>Implementació producte i validació</td>
<td>I2CAT</td>
<td>Codi i informe</td>
<td>Privat</td>
<td>M6</td>
<td></td>
</tr>
<tr>
<td>D4.3</td>
<td>Implementació producte millorat</td>
<td>I2CAT</td>
<td>Codi i informe</td>
<td>Privat</td>
<td>M8</td>
<td></td>
</tr>
<tr>
<td>D4.4</td>
<td>Mòdul PFC1</td>
<td>UPC</td>
<td>Codi i informe</td>
<td>Privat</td>
<td>M10</td>
<td></td>
</tr>
<tr>
<td>D4.5</td>
<td>Mòdul PFC2</td>
<td>UPC</td>
<td>Codi i informe</td>
<td>Privat</td>
<td>M10</td>
<td></td>
</tr>
</tbody>
</table>

### Comentaris
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nov</td>
<td>Ene</td>
</tr>
<tr>
<td>PT1. COORDINACIÓ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1.1 Reunió Inicial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1.2 Reunions plenàries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1.3 Control de progrés</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1.4 Qualitat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1.1 Pla de treball</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1.2 Presentació projecte</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1.3 Report final</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT2. ANALISI I ESPECIFICACIÓ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.1 Requeriments d'usuari</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.2 Analisi de requeriments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2.3 Especificació</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D1.1 Document de requeriments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2.2 Especificacions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT3. DISSENY I ARQUITECTURA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3.1 Disseny interfíce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3.2 Disseny i arquitectura</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3.1 Disseny d’interfície</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3.2 Arquitectura del sistema</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT4. IMPLEMENTACIÓ I VALIDACIÓ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4.1 Divisió en fases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4.2 Implementació</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4.3 Validació</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11.6 Preliminary Report

We present here the preliminary report of this project, written in April of 2011.
Chapter 12

Survey of Natural Language Interfaces to Ontologies

This survey of Natural Language Interfaces to Ontologies was written at the start of this project to explore the state of the art about this topic. One of the initial features of the visualizer was a natural language search box for the Pat.mapa semantic database, which was discarded because of time constraints.

12.1 Abstract

As the semantic web grows, an easy way to access this information is needed for end users which do not have knowledge of ontology query languages. Natural language interfaces to ontologies offer end-users a familiar and convenient option for querying ontology-based knowledge bases. In this paper, we present eleven systems that have been developed to fulfill this need. We introduce the reader to the field of the semantic web and ontologies; present the research challenges that exist on the field; review each of the eleven studied systems presenting their architecture, features and performance, and classifying them by their common characteristics; and finally we provide some insight on were this field is heading and its future challenges.

12.2 Introduction

The Semantic web is a group of methods and technologies to allow machines to understand the meaning - or “semantics” - of information on the World Wide Web. This concept was coined in the arisal of the Web 2.0, when experts of the field tried to see where the web technology will go in the future. The way to represent this information is through ontologies, which are a formal representation of the knowledge by a set of concepts within a domain and the relationships between those concepts. As the W3C started developing the formats for the semantic web, like the Resource Description Framework (RDF) in 2004, or the Web Ontology Language (OWL), efforts to store human knowledge in a machine-understandable format have increased, and thus different methods to interact with this knowledge have arisen. In this survey paper, we present
systems that serve as a natural language interface to these knowledge bases, trying to tie the
need of end-users to access these information with its availability through the Semantic Web.

As of today, the majority of information stored in ontologies is only accessible through complex
query language operations, such as the ones defined by SPARQL or other ontology query lan-
guages. In order to have this information available to users without knowledge about these query
languages, we need a simpler, more “humane” approach to query them, as it has happened in
the web with the advent of keyword-based search engines. Since in the field of the semantic web,
semantic information about the stored knowledge is on the hands of the interface engine, it makes
sense to develop a natural language interface exploiting this semantic metadata to extract the
meaning of the questions presented to the system. In this way, an useful answer to the end user
can be presented with minimum effort on his side, by only having to use the same language he
does to communicate with others. This natural language interfaces are as useful in a traditional
context (textual queries through a computer) as they can be in one were voice is used as a major
interface to the system (using a speech-to text translation), as the increasingly popular mobile
devices like smartphones and tablets are starting to be.

The systems presented in this paper are related to the natural language interfaces to databases
that have been developed in previous years. CHAT-80 [16] was a system developed in the early
eighties that transformed English questions into Prolog expressions, which were evaluated against
the Prolog database; and that served as the basis of other NLIDB systems like MASQUE [12].
Later on, TEAM [4] was developed as a NLIDB system with portability in mind, so that it could
be easily configurable by database administrations to act as an interface to their database. Many
other similar systems were developed in the eighties, like ASK [13], JANUS [17], or DATALOG [5],
each having its particularities and solving different problems in the field, but the interest in
this systems fell by the middle of the nineties, as the databases became more complex and so
its information not being as useful in a direct way to end users. However, the basis of their
development was used as inspiration for the systems introduced in this paper, as can be seen in
the references made to some of them in these articles.

The conferences where these papers have been presented, and where this field is currently dis-
cussed, are the International Semantic Web Conference (ISWC), the Annual Meeting of the
Association for Computational Linguistics, the Conference on Intelligent Text Processing and
Computational Linguistics, the International Conference on Language Resources and Evalua-
tion (LREC), the Extended Semantic Web Conference (ESWC), the Semantics, Knowledge and
Grids Conference (SKG), the International Symposium on Natural Language Processing, and
the International Conference on Natural Language Processing and Knowledge Engineering.

In this paper, we intend to summarise the current state of the natural language interface systems
to consult ontologies. We first introduce some terminology to acquaint the reader to the jargon
of the field; afterwards, we present the research challenges existing on the field, and the different
ways each paper tries to solve them, analysing their approach taking into account the translation
type, the user feedback and the learning process used; following this, we explain the guidelines
we will follow to classify and analyse each paper, and present the summary of each of the eleven
studied papers according to this system; and finally, we extract the conclusions that arise from
the study of these, trying to show where will this field lead us in the future.
12.3 Terminology

We introduce some terminology of this field to help the understanding of the survey:

**Natural Language**: A human written or spoken language, as opposed to a computer language. Due to its natural evolution and general use, it lacks the definition and unambiguousness of computer languages.

**Ontology**: a formal representation of the knowledge of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to describe the domain. They are used in artificial intelligence, the Semantic Web, systems engineering, software engineering, and more fields. They are formed by classes, attributes of these classes, relations between classes, and instances of these classes; and they follow a triplet structure: subject-predicate-object (explained below). The more common ontology languages used to describe them are OWL, RDF and F-Logic. Some known ontologies are Dublin Core, DOLCE, Wordnet, or CIDOC CRM.

**SPARQL**: SPARQL stands for SPARQL Protocol and RDF Query Language, it is an RDF query language to query ontologies in RDF. A query consists of triple patterns, conjunctions, disjunctions, and optional patterns. Example for question: “What are all the country capitals in Africa?”:

```
PREFIX abc: <http://example.com/exampleOntology#>
SELECT ?capital ?country
WHERE {
  ?x abc:cityname ?capital ;
  abc:isCapitalOf ?y .
  ?y abc:countryname ?country ;
  abc:isInContinent abc:Africa .
}
```

**Triplet**: A triple is the basic element of an ontology, consisting of a subject, a predicate, and an object, such as “Bob is 35” or “Bob knows Fred”.

**RDF**: The Resource Description Framework, a metadata data model used to describe ontologies. The predominant query language for RDF graphs is SPARQL.

**WordNet**: lexical database for the English language, which groups English words into sets of synonyms called synsets, to support automatic text analysis and artificial intelligence applications. As of 2006, the database contains 155,287 words organized in 117,659 synsets for a total of 206,941 word-sense pairs.

12.4 Research challenges

The main goal of all of these papers is creating a natural language interface for accessing knowledge stored in ontologies. This is desired because even though the resources of the semantic web are growing each day, the way to access them as an end user keeps being complicated, having to learn a query language such as SPARQL or RDQL. By creating a way to query these ontologies with natural language, all this knowledge is made accessible to end users.

Since we would want access to an arbitrary number of different knowledge bases, a domain-independent system is needed, because having to develop or adapt a system for any ontology is
not feasible. Most of the papers selected solve this problem by integrating the ontology domain information in an automatic way, apart from Aqualog, that has to be configured manually in a minimal way by the configuration files, and Orakel, that uses the Frame Mapper tool to ask the user about the mapping of verbs with relations in the ontology.

The way to accomplish this translation is different in each paper, but they share common characteristics and thus they can be grouped by their Translation type:

**Grammatical**: ACE, Aqualog, Ginseng and Orakel use grammars with general rules and domain-dependent rules from the ontology to parse and convert the question into an ontology language query. The intermediate representation of the query is usually a tree, where the words in it are related to the concepts in the ontology.

**Heuristic**: Querix, NLP-Reduce, Nokia NQ, QuestIO, FREyA, Panto and NLION use different heuristic methods to match the concepts in the question to the ones in the ontology. These methods normally consist of concept matching using the ontology concepts augmented with synsets in a single or multipass scheme.

Since the translation of the natural language question into an ontology query may not always be perfect, some systems have implemented disambiguation schemes via user interaction, like QuestIO, where the possible matchings are ranked and then presented to the user, or Aqualog or FREyA, were individual concept matchings are asked to the user if they are not unambiguous. Ginseng is a special case in this topic, because its interface consists of text entry via guided input, and so it resolved ambiguity by not letting the user input questions that will not be answered in an unambiguous way.

Learning from the user disambiguations to help answering future queries is also interesting in these systems, in order to overcome the non exact mappings between user questions and ontology queries due to the natural language implied complexity, self reference and ambiguity. This is implemented in FREyA and Aqualog, using the disambiguated terms by the user and taking the context into account to disambiguate future queries in an automatic way; which makes the system’s performance better as more questions are processed. Orakel also enables a user-assisted learning process, where non answered queries can be logged and the concepts not correctly matched shown to the lexicon builder to extend the lexicon to cover these terms.

Finally, another way to control the user’s input to ensure that an answer will be found, is by restricting the input language to a set smaller than the full English language. The ACE system does this by using the Attempto Controlled English language as its input, which is a variation of English were there are no ambiguities in the relation between subjects and objects in the sentences. Another way of doing it by guiding the user’s input to avoid ambiguity, like Ginseng uses, as explained above.

### 12.5 Classification of the papers

Taking into account was has been explained in part 3, we will classify the papers by the characteristics that arise from that development, which are the following ones:

**Translation type**: The query issued by the user needs to be interpreted in terms of the ontology concepts and translated to a formal query that can be executed on the ontology. This translation may be performed using a natural language grammar to translate the user query into an ontology query, or by using heuristic matching to match ontology concepts to concepts in the
question.

**Intermediate representation:** The intermediate representation used for the query while in the translation process. This structure gives us information about the translation process. Examples range from trees, set of triples, graphs, or set of concepts.

**Domain knowledge:** The information about the domain that the ontology covers has to be integrated in the system to correctly understand the users' queries. This domain knowledge can be used in the system in a manual way, like it was done in the first systems; manually but assisted by the user, like in the Orakel system; or in an automatic way. The manual way has a greater cost associated with it, which is the time and knowledge the integrator invests on it, but may lead to better results; while the automatic way is faster and cheaper, but can lead to imprecisions if not well done, and it depends on the semantic richness of the ontology.

**User feedback:** The systems can interact with the user in the process of the translation of the query or while presenting the result, for various reasons. Some systems use user feedback for disambiguation of concepts in the questions, or refinement of the answers given multiple choices, and this user feedback can also be further used for learning.

**Learning:** The user feedback can be used to make the system learn from previous queries, refining its answers based on the context of the question. Here we state the type of learning that the system uses, if available.

**Input language:** The input language of the system, the one the user uses to write the query. It can be full English, or a controlled version of it, or even a guided input approach that limits the vocabulary.

We will summarise the systems by giving the basics of its architecture, and by showing its interesting features that differentiate them from the rest. We also show how they were evaluated, giving details on the tests used (questions, precision and recall), and finally we show a table with all this data for easy comparison. The papers are presented in chronological order, to get a grasp of the evolution of the topic.

### 12.6 Reviews of the papers

#### 12.6.1 Querying Ontologies: A Controlled English Interface for End-Users

**Translation type:** Grammatical, rules extracted from the ontology's vocabulary and general rules

**Intermediate representation:** Discourse Representation Structure -> rules based representation

**Domain knowledge:** manual

**User feedback?:** no

**Learning?**: no

**Input language:** Attempto Controlled English
The ACE paper, by Abraham Bernstein, Esther Kaufmann and others, introduces us to a controlled English interface to ontologies using the Attempto Controlled English (ACE) language. This system works by parsing the query with the Attempto Parsing Engine, which converts it into a discourse representation structure; rewriting it with the Rewriting Grammar, (generated from rules of the domain model and general purpose rules), which produces a triplet-style PQL-Query; and finally querying the ontology with the aforementioned query.

Features

This system has two interesting parts, which are explained below:

**Attempto Controlled English**

The Attempto Controlled English is a controlled natural language and a subset of the English language. It is specified by a set of construction rules, which allow users to build simple, composite sentences, and queries, and interpretation rules, which eliminate syntactic and semantic ambiguities.

The queries done to the system are translated into a discourse representation structure by the Attempto Parsing Engine, that is implemented in Prolog as a Definite Clause Grammar. The DRSs consist of discourse referents, and conditions for them. Here is an example of one translation:

<table>
<thead>
<tr>
<th>DRS</th>
<th>First-order Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>$\exists A \cdot \exists B \cdot \text{customer}(A) \land \text{book}(B) \land \text{buy}(A, B)$</td>
</tr>
</tbody>
</table>

**Rewriting grammar**

The Rewriting grammar converts the DRs into PQL queries using two types of rewriting rules, ontology-model specific keyword rules, and general vocabulary rules. These rules are stated in the following way:

Ontology-Model Specific Keyword Rules: These rules apply if one of the keywords of the ontology (or a synonym) appears in the DRS. If it does, then it is translated into a part of a PQL query. For example, if the ontology has a property called HAS-PORT, having the query “Which process has a port” will fire one of this rules giving the PQL query “?process < #has-port > ?port”. 

50
General Vocabulary rules: The parts of the DSR not handled by the ontology specific rules are processed by the general vocabulary ones. These rules fire depending on if the query is a simple or complex sentence, of its complement or adjunct clauses, and so on. The full rule tree of the grammar can be seen here, relating to the query “Which service shows the menus of the campus restaurants over the internet?”

After these rules are applied, some post processing is done to simplify the query, via the post processing rules.

Testing

This system was tested by 20 end-users, which were asked to phrase ACE queries (after receiving a short document explaining it). 30 queries were generated, and when ran in the system (after slight manual ACE corrections), they achieved a precision of 100% and a recall of 90%.

The authors also compared the system with an SQL based one by making the end-users rephrase the queries in both languages, and found out that overall, people preferred ACE over SQL; and also tried more complex questions, receiving satisfactory answers, which speaks of the potential of the system.

12.6.2 Aqualog

Translation type: grammatical, using JAPE grammars and the RSS
Intermediate representation: triple-based data model
Domain knowledge: yes, automatic and manual in the configuration files
User feedback?: Yes, for disambiguation
Learning?: Yes, from the users’ clarification
Input language: English
System architecture

AquaLog [1], by Vanessa Lopez, Michele Pasin and Enrico Motta, is a portable question-answering system which takes queries expressed in natural language and an ontology as input, and returns answers drawn from one or more knowledge bases. Its architecture goes as follows:

A query made by the user is transformed by the Linguistic Component to a set of triples, using GATE [2] libraries and other semantic modules, like the JAPE grammars. It is then processed by the Relation Similarity service (RSS), which converts it to a query to the ontology(es) using the Class Similarity Service, string pattern libraries, the WordNet thesaurus, and other RSS modules. This ontology query is processed by the knowledge base and it is finally presented to the user as the answer to his question. All this process is being helped by the user’s feedback in case of ambiguity, which is further used to expand the vocabulary by learning from it.

Features

The Linguistic Component and the Relation Similarity Service (RSS) are the two central components of AquaLog, and together with the Learning Mechanism, they form the defining characteristics of this system. We present these features below.

Linguistic Component

The Linguistic Component’s task is to translate from NL to the triple format used to query the ontology. This component first uses the GATE processing resources for tokenizing, splitting, POS tagging, and VP chunking the query. Afterwards, a set of JAPE grammars (offered by GATE) are used to identify the intermediate representation that need to be used from 14 question types. This component is prepared to work with basic queries, basic queries with clauses (queries with modifiers, which have to be disambiguated using the ontology or user feedback), and combination of queries (which can be conjunctions or conditional queries). Some query->triple pairing can be seen as an example in img5:
Relation Similarity Service
The RSS is used to process the triples generated by the Linguistic Component: it tries to make sense of the input query by looking at the structure of the ontology and the information stored in the target ontology(es), as well as using string similarity matching, generic lexical resources such as WordNet, and a domain-dependent lexicon obtained through the use of a Learning Mechanism. This process is interactive, because the user is asked whenever an unsolved ambiguity arises. The relation and concept names are mapped from the ontology using the Class Similarity Service, which is based on several string distance metrics using an external API from Carnegie Mellon University [?].

Learning Mechanism

When an user is asked about an ambiguity that has arisen in the processing of their query, its answer is recorded in a database together with a series of constraints (the arguments of the
question, the name of the ontology and the user information) that will determine its reuse within similar contexts. These answers are later used in following user questions if they adapt to the current context, which is calculated by generalization algorithms depending on the topic of the question and the user asking it.

Testing

Aqualog has been thoroughly tested, and can be evaluated by the reader here: http://aqualog.open.ac.uk:8080/aqualog/index.html, or by downloading the source code from their repository: http://technologies.kmi.open.ac.uk/aqualog/aqualog/libraries.html.

In their tests they found five kinds of failures that a query may find:

Linguistic failures, when the NLP component is unable to generate the intermediate representation;

Data model failure, when the NL query is too complicated for the intermediate representation;

RSS failure, when the Relation Similarity Service is unable to map an intermediate representation to the correct ontology-compliant logical expression;

Conceptual failure, when the ontology does not cover the query, and

Service failure, when there is a lack of appropriate services defined over the ontology.

From the 69 questions they used to test the system, here are the results they found:

<table>
<thead>
<tr>
<th>Aqualog Evaluation</th>
<th>Aqualog success in creating a valid, Onto-Triple, or if it fails to complete the Onto-Triple is due to an ontology conceptual failure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total number of successful queries</td>
</tr>
<tr>
<td></td>
<td>Total number of successful queries without conceptual failure</td>
</tr>
<tr>
<td></td>
<td>Total number of queries with only conceptual failure</td>
</tr>
<tr>
<td>Aqualog failures</td>
<td>Total number failures (same query can fail for more than one reason)</td>
</tr>
<tr>
<td></td>
<td>RSS failure</td>
</tr>
<tr>
<td></td>
<td>Service failure</td>
</tr>
<tr>
<td></td>
<td>Data Model failure</td>
</tr>
<tr>
<td></td>
<td>Linguistic failure</td>
</tr>
<tr>
<td>Aqualog easily reformulated questions</td>
<td>Total num queries easily reformulated</td>
</tr>
<tr>
<td></td>
<td>With conceptual failure</td>
</tr>
<tr>
<td></td>
<td>No conceptual failure but no populated</td>
</tr>
</tbody>
</table>

Bold values represent the final results (%).

They also tested the learning mechanism by running the same questions with and without this component, and over several iterations. Here are the results:

<table>
<thead>
<tr>
<th>Learning mechanism performance</th>
<th>Number of queries (total 45)</th>
<th>Without the LM</th>
<th>LM first iteration (from scratch)</th>
<th>LM second iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 iterations with user</td>
<td>37.77% (17 of 45)</td>
<td>40% (18 of 45)</td>
<td>40% (18 of 45)</td>
<td>64.44% (29 of 45)</td>
</tr>
<tr>
<td>1 iteration with user</td>
<td>35.55% (16 of 45)</td>
<td>35.55% (16 of 45)</td>
<td>35.55% (16 of 45)</td>
<td>35.55% (16 of 45)</td>
</tr>
<tr>
<td>2 iterations with user</td>
<td>26% (9 of 45)</td>
<td>26.5% (9 of 45)</td>
<td>0% (0 of 45)</td>
<td>0% (0 of 45)</td>
</tr>
<tr>
<td>3 iterations with user</td>
<td>6.66% (3 of 45)</td>
<td>6.66% (2 of 45)</td>
<td>0% (0 of 45)</td>
<td>0% (0 of 45)</td>
</tr>
</tbody>
</table>

12.6.3 Ginseng

Translation type: partially dynamically generated multi-level grammar
System architecture

Ginseng [?], by Abraham Bernstein, Esther Kaufmann and Christian Kaiser, is a guided input natural language search engine for the Semantic Web. It works through a simple querying grammar which is dynamically extended based on the ontology structure, and then used to parse the queries. These queries are incrementally parsed in order to propose possible continuations of the query. Here is a visual example of the query input process:

The only vocabulary is the one defined by the loaded ontologies, complemented with synonyms. The guided way of inputting a query is similar to the autocomplete features in CLIs.

Once a query is fully entered the system uses additional query construction information to translate it into a RDF Data Query Language (RDQL)\(^1\) query and then executes it. Therefore, the Ginseng system has three parts: a partially dynamically generated multi-level grammar, an incremental parser, and an ontology-access layer. The last one is done using the Jena system\(^2\). The other two parts are explained below:

Features

Partially dynamically generated multi-level grammar

When an ontology is loaded, its dependent rules are generated to extend the static part of the grammar, which contains the ontology-independent rules specifying general sentence structures. These grammar rules are divided in two parts, the static ones, which provide the basic sentence structures and phrases for questions, be them yes/no, cardinality, conjunctive or disjunctive ones; and the dynamic ones, which are generated from the ontologies by parsing the classes, instances, objects and data properties. Here is an example of one of the grammars generated by Ginseng, where the terminals of the rules are the entries suggested in the guided input process, and the information following the | signs is the one used to create the RDQL queries:

\(^1\)http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
\(^2\)http://jena.sourceforge.net/
Incremental parser
The grammar rules presented above are used by the incremental parser in two ways: they specify the complete set of parsable sentences, which can be used to provide the user with alternatives during entry and prevent incorrect entries, and they provide the information on how to construct the RDQL queries. Therefore, the incremental parser works in the guided input section, and also in the query generation part.

Testing
To test the Ginseng system the developers ran all the 880 queries from the geographical Mooney knowledge base, and found that, out of the 40% of the queries that could be executed without modification had a precision of 92.8% and a recall of 98.4%. They were only able to test 40% of the questions due to the guided approach of the query entering system, which restricts the input domain. They also ran usability tests comparing Ginseng and SQL with IT students, and found that Ginseng was quicker and easier to use than SQL.

12.6.4 Querix
Translation type: with heuristic patterns on the syntax tree
Intermediate representation: syntax tree, triples
Domain knowledge: yes, in the matching center
User feedback?: yes, final disambiguation
Learning?: no
Input language: English

System architecture
Querix [?], by Esther Kaufmann, Abraham Bernstein and Renato Zumstein, is a domain-independent natural language interface that uses clarification dialogs to query ontologies. The
system consists of seven parts: a user interface, an ontology manager, a query analyzer, a matching center, a query generator, a dialog component, and an ontology access layer. These parts are presented in the “features” section below.

Features

Ontology Manager: enhances the resources of the ontology with synonyms from WordNet.

Query Analyzer
This part of the system uses the Stanford Parser to build a syntax tree of the query, which is then used to build a query skeleton using these categories: Noun (N), Verb (V), Preposition (P), Wh-Word (Q), and Conjunction (C). These skeleton is also augmented with synonyms from WordNet.

Matching Center
It is the core component, it tries to match the skeleton with the ontology triples in a three step process:
1. Matches against a small set of heuristic patterns, so that it identifies subject-property-object patterns of a query. They overlap in their first or last word category to enable the joining of the triples in step 3.
2. Searches for all matches between the synonym-enhanced nouns and verbs of the input query with the resources and their synonyms in the ontology.
3. Matches the triple patterns from step 1 and the resources from step 2, taking into account the domain and range information of the ontology.

Query generator: It composes a ranked list of SPARQL queries from the joined triples. using the cost function of the matching steps.

Dialog component: If two SPARQL queries have the same cost, this component asks the user for the answer he wants with a dialog box.

Ontology access layer: To execute the generated SPARQL query, Querix uses Jena as ontology access layer and the Pellet reasoner. It then presents the answer to the user.

Testing
The Querix system was tested with 215 of the 879 queries of the Mooney database, being this subset the syntactic representation of the total. Of these questions, after removing 21 of them due to problems in the parsing, its semantics, or mistakes in the ontology, a precision of 87,11% and a recall of 86,08% was obtained. This shows, according to the authors, that Querix, even though depending on the quality of the vocabulary of the ontology, shows good results while having the advantage of being completely portable.
12.6.5 NLP-Reduce

Translation type: “naïve”, matching triples with words recursively
Intermediate representation: triples
Domain knowledge: yes, for the query generator
User feedback?: no
Learning?: no
Input language: no

System architecture

NLP-Reduce [?] by Esther Kaufmann, Abraham Bernstein and Lorenz Fischer, is a naive but domain-independent natural language interface for querying Semantic Web knowledge bases (KBs). It consists of five main parts: a user interface, a lexicon, an input query processor, a SPARQL query generator, and an ontology access layer. All of them except the query generator are pretty straightforward: the lexicon uses WordNet for synonyms of the ontology terms and the Porter Stemmer for stemming them; and the input query processor removes stop words and punctuation marks. The main part of the system is the query generator, which is explained below:

Features

Query Generator

The query generator matches the query words to the synonym-enhanced triples stored in the lexicon and generates SPARQL queries for the matches. It does this on a four step process:
1. Searches for triples in the lexicon in which at least one of the query words occurs within the label of an object property. It then ranks the found triples favoring those that cover more words and those that have words which stems show a better agreement with the query words over others.
2. Searches for properties in the lexicon that can be joined with the triples found in step 1 by the remaining query words taking domain and range information into consideration. Then they are combined with the result of step 1.
3. Searches for all datatype property values that match the remaining words of the query.
4. Generates the corresponding SPARQL query for the join of the retrieved triples that achieved the highest scores in steps 1 to 3. It also removes semantically equivalent duplicates and passes the SPARQL query to the ontology access layer.

Testing

NLP-Reduce was tested with two Mooney knowledge bases:
251 questions of the restaurant KB, 192 answered (94.6%), 67.7% precision, 69.6% recall
879 questions of the geography KB, 308 answered (66%), 70.7% precision, 76.4% recall
The precision an recall were calculated in an strict matter, where if the answer was not exactly what should be, scored 0%. They arrive to conclusions similar to the ones of the Querix paper.
12.6.6 Nokia NQ

**Translation type:** multipass refinement using language tags  
**Intermediate representation:** Abstract and concrete (subgraph) semantic interpretation  
**Domain knowledge:** yes, for the semantic tagging  
**User feedback?** no  
**Learning?** no  
**Input language:** English

**System architecture**

The Nokia NQ system was developed by Alexander Ran and Raimondas Lencevicius with an interest in using language based interaction on mobile devices, and so it was optimized to increase portability, limit computational complexity of the language components and maximally automate the process of integration between language systems and databases. The system processes a natural language question in five stages: semantic tagging, parsing, abstract semantic interpretation, concrete semantic interpretation and heuristic ranking.

**Features**

**Semantic tagging**  
It performs two functions:  
value tagging: marks tokens in the question that correspond to values in the database, and includes recognition of expressions for regular ordered value types such as numbers, time, and date.  

**Category tagging**  
Identifies potential references to database entities such as classes and properties, using “language tags”, which are linguistic labels attached to the database elements to generate information about terms that might be used to refer to the database items.

**Parsing**  
The parser has no domain dependency. It parses the question and process it so that taggings that do not produce a parse are rejected; alternative parses are ranked (if a probabilistic model is available); and the focus of the question and most probable attachment of the phrases are identified.

**Abstract semantic interpretation**  
With the information added by the semantic tagging and parsing of the question, an abstract meaning representation is generated using binary infix predicates. This representation does not use the same property names as the database, so it requires further interpretation by processing the “semantically related” entities: if we see the ontology as a graph, for two nodes n1 and n2 of it, they are semantically related over selected class and predicate domain D if there exists a path of predicates from D that connects these nodes.

**Concrete semantic interpretation**  
This section of the system converts the abstract meaning representation to a formal database query by adding information about the organization of the database. It does this by generating a set of possible concrete semantic interpretations gotten through searching the database graph identified by the abstract meaning representation.
Heuristic ranking

Finally, the set of subgraphs is ranked in terms of suitability as a concrete semantic interpretation of the question. It is done in a domain independent way via: the semantic relatedness rule (introduced before); the active voice rule, where the language tags describing a relation are required to be used with active voice of verbs; and taking into account the weight of retrieved subgraphs, which negatively correlate with their appropriateness as semantic interpretation of the question.

Testing

The NQ system was tested over different semantic repositories, including personal information databases, intranet corporate phonebooks, the CIA World Factbook, and a job search repository. The most extensive experiment was in this last one, where 200 questions were executed against the system and returned precision and recall close to 100%.

12.6.7 ORAKEL

Translation type: Grammatical, using Logical Description Grammars
Intermediate representation: LDG trees
Domain knowledge: manual, with the FrameMapper tool
User feedback?: no
Learning?: manual using feedback from wrong questions and FrameMapper
Input language: English

System architecture

Orakel [?], by Philipp Cimiano, Peter Haase and others, is a natural language interface to knowledge bases, using a lexicon built by hand for the ontology in use, and using an inference engine to provide answers not included in the KB but that can be inferred from them. The system architecture is the following one:
In the case of an end user, he asks a question that is semantically interpreted by the Query Interpreter, which parses it and constructs the query in a first order logic logical form, using the domain-independent lexicon (from the DOLCE ontology) and the one provided by user input from the FrameMapper. This form is then converted to the knowledge base query language by the Query converter, using a Prolog description of it. Finally, the answer generation component evaluates the query with the KB and returns the answer to the user.

In the case of the lexicon engineer, he uses the FrameMapper interface to the KB, which supports him in specifying by graphical means the mapping from language to relational predicates defined in the knowledge base. This interaction generates the domain lexicon specific for the application in question.

Features

Query Interpreter

ORAKEL uses Logical Description Grammars (LDG) as the basis of their system, which are trees consisting of nodes labeled with syntactic information. The nodes of these trees can be marked negatively if they correspond to arguments which need to be inserted, or positively if they denote variables to be inserted as an argument. The operation applied to these trees consists of identifying positively with negatively marked nodes with each other within one or across trees. The nodes are matched if their feature value pairs match, which are the head, genus, function and type. The verbs are then represented by subcategorization frames, which is the verb and the arguments needed for it; the names, pronouns and prepositions correspond to elementary trees, and the adverbs and adjectives to trees with them in one leaf and their corresponding matching category in the other.

The parser used identifies positively and negatively marked nodes respecting: the syntactic category of nodes, feature values, ontological constraints, surface word order, and syntactic
dominance relations. It is an Early-type bottom-up parser using top-down information, the full pseudocode of it can be seen in the paper.

Finally, the logical formula of the input question is constructed using lambda calculus, with an extended compositional operator. Their implementation can be seen at www.cimiano.de/orakel. For treating inference, inferencing rules can also be added to the ontology to infer implicit relations, like stating that a river x flows through a location z if x flows through y and y is located in z:

\[ \forall x, z. \text{flowthrough}(x, z) < - \exists y. \text{flowthrough}(x, y) \land \text{location}(y, z). \]

**Query Converter**

The first order logic query obtained by the query interpreter is translated into the logical language of the database using a translation component specified declaratively in Prolog. Therefore, a query can be converted into different query languages (F-Logic for the Ontobroker system, SPARQL for OWL) depending on the translator, as long as the query language provides extra-logical predicates for counting and for numerical comparisons, and its reasonably expressive.

**FrameMapper**

Part of the domain-dependent lexicon is generated automatically by reading all the concepts and instances of the ontology and generating grammar trees representing them using their labels. The other part is generated by the user with the FrameMapper tool, by mapping verbs, adjectives and relational nouns to corresponding relations specified in the domain ontology. This is done by instantiating subcategorization frames and mapping these to domain-specific relations in the ontology. Examples of these mappings can be seen below:

- location(pcomp(of): x) $\rightarrow$ locatedIn(x: location, y: location)
- live(subj: x,pobj(in): y) $\rightarrow$ inhabitants(y: state/city, x: integer)
- capital(pcomp(of): x) $\rightarrow$ capital(x: city, y: state)
- pass(subj: x, pobj(through): y) $\rightarrow$ flow through(x: river, y: city)
- pass(subj: x, pobj(through): y) $\rightarrow$ located at highway(y: city, x: highway)
- border(subj: x, obj: y) $\rightarrow$ borders(x: location, y: location)
- adj(big,bigger,biggest,positive) $\rightarrow$ inhabitants(city, integer)

These mappings are crucial to ORAKEL, since it does not use simple heuristics based on matches between relation names and verbs or nouns like other systems, opting for the user input for these mappings instead.

**Testing**

To evaluate the system, the authors provided three different tests:

1. Test with a geography KB, using a lexicon built by the author, and other two lexicons built by end-users (in two iterations, using the failed questions for feedback). A total of 454 questions were asked to the system by end-users, with a precision of about 82% and a recall of about 48%. The question types were also analysed, and they found that the system had a coverage of 93%, and that they could add the question types not covered to the system.

2. Runtime evaluation, depending on the length of the question. The time taken for answering a query went from 0.01s for 3 word questions to a max of 0.56s for 13 word questions. Therefore it was concluded that ORAKEL can answer questions in real time.

3. Real-world application of the system, by adding a natural language interface to BT’s digital library described by the PROTON ontology (http://proton.semanticweb.org/), which was much bigger than the one used in the previous experiment. The lexicon was created using FrameMapper
in 6 hours, and after three iterations, the questions asked by end users got a precision of 73% and a recall of 61%.

12.6.8 Panto

Translation type: NPs to QueryTripes to OntoTriples
Intermediate representation: QueryTriples and OntoTriples
Domain knowledge: in the lexicon
User feedback?: no
Learning?: no

Input language: English

System architecture

Panto [?], by Chong Wang, Miao Xiong and others, is a Portable nAtural laNguage inTerface to Ontologies. Its system architecture is the following one:

It consists of two parts, the Ontology processing, where the lexicon is built (composed of the ontology entities (classes, properties and instances), general dictionaries (using WordNet synsets), and user defined synonyms); and the Query Processing, which translates natural language questions into SPARQL queries. This is done by parsing the question with the Stanford parser, and then sending the parse tree to the core processing engine of the system, the Translator. Here is where the parsed queries are converted into SPARQL, and will be explained in more detail below.

Features

Translator
The translator works on the basis that nominal phrases (phrases consisting of a subject, verb,
and object) can be translated into triples that follow the ontology structure. The full translation process can be visualized here:

1. QueryTriple Extractor
   This section extracts the nominal-phrase pairs of the parse tree to form QueryTriples. It does it in the following steps: First, since the nominal phrases (NPs) can be nested, it identifies the baseNPs of the query, and then identifies and propagates the related NPs following the rules stated by Michael Collins [?]. Second, it links the BaseNPs with one another where there is modification relationship to form BaseNP pairs. The two BaseNPs in such a pair, together with the words which syntactically connect them, form a QueryTriple. Finally, it specifies the internal structure for QueryTriple, separating its subject and object into a structure such as [pre-modifier . head noun . post-modifier].

2. OntoTriple Extractor
   Here the QueryTriples are mapped to the OntoTriples that correspond in the ontology. It does this by mapping the QueryTriples subjects and objects to entities of the ontology using the lexicon using semantic (synonyms) and morphological (string metrics or heuristic rules) matching; and then uses this information to match each QueryTriple to one or more OntoTriples (triples that are compatible with some statements in the ontology), depending on its type.

3. Target and Modifier Extractor
   The parameters for the SELECT part of the SPARQL query are extracted here, by finding the wh-word or an imperative verb of the question (“list”, “give me”, etc), and using the nouns in the same or the directly followed constituent as targets. The FILTER part, if necessary, is also produced here, by processing the modifier indicators, which are negating, superlative, comparative or conjunctive/disjunctive words in the query.

4. SPARQL Generator
   Here the targets extracted in the previous phase are related to the OntoEntities and OntoTriples for the SELECT; the OntoTriples are directly interpreted as triple patterns for the WHERE clause, making instance and property entities as URIs in the ontology, and class entities or RDF Literal Type entities as variables; and the FILTER clause is created using the information from step 3 depending on the indicator type. With all the SPARQL query parts created, the query is finished.
Testing

The authors tested Panto with the Mooney database with 877 geographical questions and 238 restaurant ones. They got a precision of 88.05% and 85.86%, and a recall of 90.87% and 96.94%, which gives an average precision of 85.58% and an average recall of 92.16%. They also note that Panto's coverage is about the same as Aqualog's, and that all the questions from the Mooney database were accepted by the system.

12.6.9 QuestIO

Translation type: matching concepts with relations in the ontology
Intermediate representation: ranked relations and key concepts
Domain knowledge: in the initialization process
User feedback?: no
Learning?: no
Input language: English

System architecture

QuestIO [?] is a system for querying ontologies using unconstrained language-based queries, and it is domain independent, easily embeddable and requires no end-user training. The system architecture for answering a query is shown below:

Each user query is interpreted using the Query Interpreter in the User Interface. It is then analysed by two components:

the Key Concept Identification Tool (KCIT) identifies key concepts inside the query, which refer to mentions of ontology resources such as instances, classes, properties or property values.
The Context Collector collects all words from the query that are not recognised by KCIT, but could be useful in the process of generating the formal query, which are prepositions, to be used when analysing the direction of a supposed relation between the two concepts; keyphrases like, for example “What are”, “What is” or “How many”; and chunks, which are any part of a query that is between two identified key concepts, used later in the relation ranking process.

It is then processed by the query analyzer, explained in the features section, and a SeRQL query is created dynamically. The answer retrieved by running the query is then shown to the user.
Features

Automatic domain knowledge extraction
When the system is initialized, it preprocess the ontology resources (classes, instances, properties and property values) and extract any human-understandable lexicalisations. They are then analysed by the OntoRoot Application, which tokenizes them, and assigns parts-of-speech and lemma information to each one to form a gazetter.

Query Analysers

Filtering concepts:
When identifying key concepts, more than one annotation can appear over the same token or a set of tokens, which needs to be disambiguated: this is done by giving priority to the longest matching annotations. This rule is based on the assumption that longer names usually refer to the more specific concepts or instances whereas shorter ones usually refer to more generic terms.

Identifying relations between key concepts:
The system retrieves and analyses potential relations between identified key concepts, based on the defined relations in the ontology. To retrieve these relations it uses the ontology-based reasoning provided by the reasoning component.

Ranking potential relations:
Retrieved relations are then scored using a combination of three factors: the similarity of the relation's name with the part of the query; the specificity score, calculated in relation to the position of the property in comparison to other existing properties in the ontology hierarchy; and the distance score, which reflects the position of the domain and range classes of the property inside the ontology hierarchy.
Testing

The creators of QuestIO compare it to Aqualog running 22 questions on both systems and stating which ones were correct, correct after reformulation, partially correct, and wrong:

<table>
<thead>
<tr>
<th></th>
<th>QuestIO</th>
<th>Correct</th>
<th>Aqualog</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>9 (56.25%)</td>
<td>56.25%</td>
<td>5 (31.25%)</td>
<td>31.25%</td>
</tr>
<tr>
<td>c. correct</td>
<td>0</td>
<td>(0%)</td>
<td>3 (18.75%)</td>
<td>18.75%</td>
</tr>
<tr>
<td>p. correct</td>
<td>5 (31.25%)</td>
<td>15.63%</td>
<td>3 (18.75%)</td>
<td>9.35%</td>
</tr>
<tr>
<td>failed</td>
<td>2 (12.5%)</td>
<td>0%</td>
<td>5 (31.25%)</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>71.88%</td>
<td></td>
<td>59.35%</td>
<td></td>
</tr>
</tbody>
</table>

They also tested the portability and scalability of the system by checking it with two overlapping knowledge bases of different sizes, where one has about 40 times more resources than the other. The questions asked had the same answer in both, since one is a subset of the other. The results in initialization time and execution time of different queries can be seen below:

<table>
<thead>
<tr>
<th>Queries</th>
<th>TG</th>
<th>KIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>countries located in Asia</td>
<td>0.547</td>
<td>5.65</td>
</tr>
<tr>
<td>capitals of countries located in Asia</td>
<td>0.203</td>
<td>5.6</td>
</tr>
<tr>
<td>capitals of countries in southern Europe</td>
<td>0.109</td>
<td>5.4</td>
</tr>
<tr>
<td>which are the political regions in Europe</td>
<td>0.141</td>
<td>11.1</td>
</tr>
<tr>
<td>is London capital of any country?</td>
<td>0.625</td>
<td>111</td>
</tr>
<tr>
<td>capital country France</td>
<td>0.344</td>
<td>10.6</td>
</tr>
</tbody>
</table>

12.6.10  NLION

Translation type: Semantic Relation Tagging
Intermediate representation: list of PCE and PPE tress
Domain knowledge: in the SRT
User feedback?: no
Learning?: no
Input language: English

System architecture

NLION [?] by Vivek Anandan Ramachandran and Ilango Krishnamurthi, is a Natural Language Interface for querying ONtologies, and uses semantic relation tagging (SRT) to convert NL queries
to SPARQL. The architecture of the system is as follows:

When the user submits a query in the form of NL to the system, it accepts and it expands it using spell checks, stemming, and words merging. Then, SRT is performed on it, and the result is passed to the SPARQL generator, which is then processed by the ontology engine and its output displayed to the user. The interesting part of this system is the Semantic Relation Tagging, which is explained below:

**Features**

**Semantic Relation Tagging and Structure Recognition**

Each word in the query is tagged by NLION with a label indicating if it is a possible concept (PCE), possible property (PPE), both or none. These labeled concepts are then structured in a set of trees and manipulated to form the triples in the SPARQL query with the following
transformations:
Concept Tagging: if two PCEs refer to the same concept, they are reformulated into a root-child relation.
PCE Fixing: the trees with PCEs on their roots are considered for further processing.
Value Tagging: if there are PPEs talking about the same property, they are reformulated so that the main property is on the root.
PPE Fixing: the PPE is reformulated to the same word that is referred in ontology, depending on its matching with each PCE.
Ambiguity resolving: the duplicated and ambiguous PCEs and PPEs are removed so that only one of each remains.

These PCEs and PPEs in tree form are finally used to construct the SPARQL query following this template:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns: <http://namespace.org/>
SELECT ?PCE_1 ......... ?PCE_m
WHERE { ?PCE_1 rdf: PPE_1 ?x_1 ..... PCE_1 rdf: PPE_1 ?x_m
       ?PCE_1 rdf: PPE_1 ?y_1 ..... PCE_m rdf: PPE_1 ?y_n.}
```

Testing

NLION was tested with a sample ontology using 67 concepts and 36 relations, and 125 simple sentences. They found that 61.60% of the queries where translated correctly by the system, and explained its moderate success because the SPARQL generated by NLION contains concepts and relations not present in the DDO.

12.6.11 FREyA

**Translation type:** Mapping of query to POCs, POCs-OCs, query
**Intermediate representation:** POCs and OCs
**Domain knowledge:** in the identification of ontology concepts
**User feedback?**: yes
**Learning?**: yes
**Input language**: English

**System architecture**

**Features**

FREyA [?]: Feedback, Refinement and Extended Vocabulary Aggregation, is a natural language interface to ontologies with learning from user feedback, based on QuestIO, introduced before, by Danica Damljanovic, Milan Agatonovic and Hamish Cunningham. Its architecture is broken down in three steps: Identification and verification of ontology concepts, Generating SPARQL, and Identification of the answer type and presenting the results to the user.
Features

Identification of Ontology Concepts
This process is shown in the image below:

The Potential Ontology Concepts (POC) are derived from the syntactic parse tree (generated by the Stanford parser) using several heuristic rules, and refer to question terms which could be linked to an ontology concept. Then they are attemptedly mapped to Ontology Concepts (OC) automatically, by comparing the text span with the ontology labels; or manually, by asking the user to choose between suggestions generated from the closest OC, and ranked with Monge Elkan\(^3\) metrics and the Soundex\(^4\) algorithm.

Generating SPARQL
The SPARQL query is generated as a set of OCs, by inserting potential joker elements in between OCs (for example if the first two are a property and a class, a joker class is added before them), and then generating the set of triples from them, taking into account the domain and range of the properties. With this, the SPARQL query is created by combining the triples based on the OC type, and adding the relevant parts to the SELECT and WHERE clauses.

Answer Type Identification
To identify the answer type of the question, the output of the syntactic parsing is combined with the ontology-based lookup coupled with several heuristic rules \(^5\). It is also presented as a graph (with the JIT library\(^5\)), with the answer type in the center, and the answer on the nearest cycle:

---

\(^3\)http://www.dcs.shef.ac.uk/~sam/stringmetrics.html#monge
\(^4\)http://en.wikipedia/wiki/Soundex
\(^5\)www.thejit.org
Learning
The system’s learning approach is inspired Reinforcement Learning (RL) Each suggestion has its initial ranking. Each time the suggestion is selected by the user, it receives a reward of +1 while all alternative ones receive -1. The system then learns to place the correct suggestion at the top for any similar questions, which are identified by a combination of a POC and the closest OC.

Testing
The authors evaluate FREyA’s correctness, ranked suggestions, and learning mechanism with 250 questions from the Mooney DB. The correctness was measured as the precision and recall of the system, which are both 92.4% (taking into account that a question may be correct after 0, 1 or 2 clarification dialogs presented to a user). The ranking algorithm was tested with the Mean Reciprocal Rank, which is the average of the reciprocal ranks (multiplicative inverse of the correct rank) of results for a sample of queries. This was calculated to be 0.81. The learning mechanism was tested by selecting 103 questions which required 1 clarification dialog, and doing training with them: they then improved the initial ranking by 6%.

12.7 Comparison table
Here we compare the systems previously reviewed based on the classification stated by section 5.
<table>
<thead>
<tr>
<th>Paper</th>
<th>Translation type</th>
<th>Use of domain knowledge?</th>
<th>Intermediate representation</th>
<th>User feedback?</th>
<th>Learning?</th>
<th>Input language</th>
<th>Auxiliary knowledge</th>
<th># queries</th>
<th>precision</th>
<th>recall</th>
<th>n or r</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Grammatical, general rules and rules based on the ontology domain</td>
<td>manual</td>
<td>discourse representation structure</td>
<td>no</td>
<td>no</td>
<td>Attempto Controlled English</td>
<td>Mooney Natural Language Learning Data (testing)</td>
<td>30</td>
<td>100,00%</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Aqualog</td>
<td>Grammatical, using JAPE grammars and the RSS</td>
<td>automatic and manual (configuration files)</td>
<td>automatic and manual (configuration files)</td>
<td>yes, disambiguation</td>
<td>yes, user disambiguation feedback</td>
<td>English</td>
<td>Gate + JAPE grammars, WordNet</td>
<td>no data</td>
<td>no data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ginseng</td>
<td>Grammatical, partially dynamically generated multi-level grammar</td>
<td>automatic</td>
<td>grammar tree</td>
<td>yes, guided input</td>
<td>no</td>
<td>English, guided input</td>
<td>Mooney DB (testing), Synonyms table (not mentioned)</td>
<td>352</td>
<td>92.80%</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Querix</td>
<td>Heuristic, with heuristic patterns on the syntax tree</td>
<td>automatic</td>
<td>syntax tree, triples</td>
<td>yes, final disambiguation</td>
<td>no</td>
<td>English</td>
<td>Stanford parser, WordNet, Mooney DB (testing)</td>
<td>194</td>
<td>87.11%</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>NLP-Reduce</td>
<td>Heuristic, “naive”, matching triples with words recursively</td>
<td>automatic</td>
<td>triples</td>
<td>no</td>
<td>no</td>
<td>English</td>
<td>Stanford parser, Porter Stemmer, Mooney DB (testing)</td>
<td>500</td>
<td>69.50%</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Nokia NQ</td>
<td>Heuristic, multi pass refinement using language tags</td>
<td>automatic</td>
<td>Abstract and concrete (sub-graph) semantic interpretation</td>
<td>no</td>
<td>no</td>
<td>English</td>
<td>not mentioned</td>
<td>200</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>--------------------------------------</td>
<td>----------------</td>
<td>-----------</td>
<td>----------------</td>
<td>---------------------</td>
<td>-----------</td>
<td>-----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuestIO</td>
<td>Heuristic, matching concepts with relations in the ontology</td>
<td>automatic</td>
<td>ranked relations and key concepts</td>
<td>no</td>
<td>no</td>
<td>English</td>
<td>GATE</td>
<td>no data</td>
<td>no data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FREyA</td>
<td>Heuristic, Mapping of query to POCs, POCs-OCs, query</td>
<td>automatic</td>
<td>POCs and OCs</td>
<td>yes, disambiguation</td>
<td>yes, user disambiguation feedback</td>
<td>English</td>
<td>GATE, OntoRoot Gazetteer, Stanford Parser, JIT</td>
<td>250</td>
<td>92.40%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panto</td>
<td>Heuristic, NPs to QueryTripes to OntoTriples</td>
<td>automatic</td>
<td>QueryTriples and OntoTriples</td>
<td>no</td>
<td>no</td>
<td>English</td>
<td>WordNet, Protege, Stanford Parser, Mooney BD (testing)</td>
<td>1115</td>
<td>85.58%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLION</td>
<td>Heuristic, Semantic Relation Tagging</td>
<td>automatic</td>
<td>list of PCE and PPE trees</td>
<td>no</td>
<td>no</td>
<td>English</td>
<td>not mentioned</td>
<td>125</td>
<td>61.60%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORAKEL</td>
<td>Grammatical, using Logical Description Grammars</td>
<td>manual, using the Frame Mapper tool</td>
<td>LDG trees</td>
<td>no</td>
<td>manual, by refining matchings with FrameMapper</td>
<td>English</td>
<td>not mentioned</td>
<td>454</td>
<td>82.00%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12.8 Conclusions

The main goal of the papers reviewed in this survey was to implement a natural language interface to knowledge bases that had most of the coverage of the traditional query languages, but with a much easier way of use, focused on end users with no knowledge of query languages. As it can be seen by the testing procedures that were followed by the authors, several of these papers arrived to the conclusion that these natural interfaces to knowledge bases are indeed both easier and quicker to use than the query languages previously used like SPARQL, which need previous knowledge by the user and also information about the ontology concepts and relations. Even more, taking into account that the majority of the papers got a pretty good (more than 80% precision and recall) performance when translating user questions, it can be stated that the main goals of the papers have been accomplished.

Taking into account the characteristics studied in this paper, we can see that these systems have evolved from manual knowledge acquisition to an automatic one, to ease up the adaptation to new knowledge bases, since they give results as good as the ones that do it manually. We can also see that systems using heuristic translation with ranking work as well as the ones using grammatical translation, so that both ways are acceptable. The language used to interact with the system has evolved from a controlled one into full compliance with the English language, and systems like Orakel try to separate the parser module from the system to make it adaptable to other languages. Finally, the use of user refinement and learning contribute to a better performance of the system as it is more used, so it is a good thing to implement to increase the correctness of returned answers.

Even though the authors of the papers presented here have accomplished a lot, there is still room for improvement. Most of these papers do not resolve problems with quantifies (how many x, number of x, is x longer than y) and queries with temporal components, like dates or relative pointers like “3 months ago”, problems that are mentioned and treated in papers like [8]. Another interesting feature that are being studied is being able to join the information from several KBs, like the Aqualog successor, PowerAqua [7] is trying to achieve. In this way, the system might be able to answer questions by combining information from several ontologies, and so broadening the knowledge contained within to more than what each ontology offers.

Apart from these enhancements, the field of natural language interfaces to ontologies seems to be broadly covered by the systems presented in this survey, so it can be speculated that future work in the field will aim at refining the precision and recall, using user feedback to further improve the system through learning, solving the missing language constructs like counting or temporal semantics, and integrating information from several knowledge sources.
Bibliography


Chapter 13

Conclusions

In this project we have developed two distinct tools that interact with the same semantic database: The Pat.mapa cultural heritage semantic database built by the Culture Department of the Generalitat de Catalunya. The Pat.mapa Extract and Review tool extracts events from texts attached to this semantic database, and lets the end user review them before adding them; while the Pat.mapa Ontology Visualizer lets the end user interact with this semantic database in an intuitive way.

As assets gained during this project, apart from the obvious benefits provided by the aforementioned software developments, is the knowledge and familiarisation our team has acquired on the field of work of the semantic web, semantic database construction, natural language processing, linked data and open data, complex data interfaces, agile web development, HTML5 technologies, and more. We are very glad of what we have accomplished, as it sets the basis for a very strong growth of our team in these emerging fields.

As future developments, we will continue improving both tools as said in section ??, and we will also pursue multiple projects related to the semantic web and complex data visualization.
Bibliography


