Complex Question Answering on Semi-Structured Repositories

A user centric process enhanced with context

José Ricardo Marques de Jesus Brandão

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Tecnologias do Conhecimento e Decisão

Orientador: Paulo Alexandre Fangueiro Oliveira Maio

Co-orientador: Nuno Alexandre Pinto da Silva

Júri:

Presidente:

Doutora Maria de Fátima Coutinho Rodrigues, Instituto Superior de Engenharia do Porto

Vogais:

Doutor António Jorge Santos Pereira, Instituto Superior de Engenharia do Porto

Doutor Paulo Alexandre Fangueiro Oliveira Maio, Instituto Superior de Engenharia do Porto

Doutor Nuno Alexandre Pinto da Silva, Instituto Superior de Engenharia do Porto

Porto, Outubro 2012

Dedication

In honor of my great friend, Leão

Resumo Alargado

A Teia Mundial (Web) foi prevista como uma rede de documentos de hipertexto interligados de forma a criar uma espaço de informação onde humanos e máquinas poderiam comunicar. No entanto, a informação contida na Web tradicional foi/é armazenada de forma não estruturada o que leva a que apenas os humanos a possam consumir convenientemente. Consequentemente, a procura de informações na Web sintáctica é uma tarefa principalmente executada pelos humanos e nesse sentido nem sempre é fácil de concretizar.

Neste contexto, tornou-se essencial a evolução para uma Web mais estruturada e mais significativa onde é dado significado bem definido à informação de forma a permitir a cooperação entre humanos e máquinas. Esta Web é usualmente referida como Web Semântica. Além disso, a Web Semântica é totalmente alcançável apenas se os dados de diferentes fontes forem ligados criando assim um repositório de Dados Abertos Ligados (LOD).

Com o aparecimento de uma nova Web de Dados (Abertos) Ligados (i.e. a Web Semântica), novas oportunidades e desafios surgiram. Pergunta Resposta (QA) sobre informação semântica é actualmente uma área de investigação activa que tenta tirar vantagens do uso das tecnologias ligadas à Web Semântica para melhorar a tarefa de responder a questões.

O principal objectivo do projecto World Search passa por explorar a Web Semântica para criar mecanismos que suportem os utilizadores de domínios de aplicação específicos a responder a questões complexas com base em dados oriundos de diferentes repositórios.

No entanto, a avaliação feita ao estado da arte permite concluir que as aplicações existentes não suportam os utilizadores na resposta a questões complexas.

Nesse sentido, o trabalho desenvolvido neste documento foca-se em estudar/desenvolver metodologias/processos que permitam ajudar os utilizadores a encontrar respostas exactas/corretas para questões complexas que não podem ser respondidas fazendo uso dos sistemas tradicionais. Tal inclui:

(i) Ultrapassar a dificuldade dos utilizadores visionarem o esquema subjacente aos repositórios de conhecimento;

(ii) Fazer a ponte entre a linguagem natural expressa pelos utilizadores e a linguagem (formal) entendível pelos repositórios;

(iii) Processar e retornar informações relevantes que respondem apropriadamente às questões dos utilizadores.

Para esse efeito, são identificadas um conjunto de funcionalidades que são consideradas necessárias para suportar o utilizador na resposta a questões complexas. É também fornecida

uma descrição formal dessas funcionalidades. A proposta é materializada num protótipo que implementa as funcionalidades previamente descritas.

As experiências realizadas com o protótipo desenvolvido demonstram que os utilizadores efectivamente beneficiam das funcionalidades apresentadas:

- Pois estas permitem que os utilizadores naveguem eficientemente sobre os repositórios de informação;
- O fosso entre as conceptualizações dos diferentes intervenientes é minimizado;
- Os utilizadores conseguem responder a questões complexas que não conseguiam responder com os sistemas tradicionais.

Em suma, este documento apresenta uma proposta que comprovadamente permite, de forma orientada pelo utilizador, responder a questões complexas em repositórios semiestruturados.

Palavras-chave: Pergunta Resposta, Questões Complexas, Dados Ligados, Web Semântica, Ontologia

Abstract

The World Wide Web (WWW) was envisioned as a network of interlinked hypertext documents thus creating an information space where humans and machines should be able to communicate. However, information published in the traditional WWW was/is unstructured and therefore is (mostly) consumable by humans only. As a consequence, searching and retrieving information in this syntactic and ever evolving WWW is a task that is mainly performed by humans and therefore it may not be trivial.

In this sense, the evolution to a more structured and meaningful web where information is given well-defined meaning thus enabling cooperation between humans and machines is mandatory. This web is usually referred to as Semantic Web. Moreover, the Semantic Web is only fully achievable if data from different resources is connected in order to create a Linked Open Data (LOD) repository.

This new Web of Linked (Open) Data (i.e. the Semantic Web) has opened a new set of opportunities but also some new challenges. Question Answering (QA) over semantic information is now an active research field that tries to take advantage of the Semantic Web technologies to improve the question answering task.

In this sense, the main goal of this work is to help users finding accurate answers for complex questions that may not be answered using traditional systems. To achieve this goal, it is proposed a user centric process comprehending a set of functionalities that are iteratively, incrementally and interactively exploited. The proposed process and functionalities aim to help users building complex queries against semi-structured repositories (e.g. LOD repositories).

Keywords: Question Answering, Complex Questions, Linked Data, Semantic Web, Ontology

Acknowledgements

I would like to thank all the people that in some way helped during this important path. However, I would like to emphasize the special care and/or assistance provided by certain people.

I would like to thank both my advisors for all the support and wisdom. It is with immense gratitude that I acknowledge the unconditional support and help of my Professor Paulo Maio; to Nuno Silva especially for his advices and constructive criticism.

I would like to thank the World Search project (QREN11495) for the opportunity of developing my MSc thesis in its context; and the OOBIAN project (QREN 12677) for providing significant knowledge that helped me elaborating this thesis.

I cannot find words to express my gratitude to my parents, José Brandão and Silvina Brandão, for all the love, care and unconditional support.

I owe my deepest gratitude to my girlfriend, Sara Silva, for helping me in the most critical moments and celebrate my accomplishments.

Finally, I thank all my colleagues at ISEP the companionship and friendship during all these years.

Table of Contents

List of Figures

List of Tables

Acronyms and Symbols

List of Acronyms

API	Application Programming Interface
DL	Description Logics
DBMS	Data Base Management Systems
GUI	Graphical User Interface
HTML	HyperText Markup Language
IR	Information Retrieval
IT	Information Technology
NLP	Natural Language Processing
OWL	Web Ontology Language
QA	Question Answering
RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
SPARQL	Simple Protocol and RDF Query Language
URI	Uniform Resource Identifier
WWW	World Wide Web
XML	eXtensible Markup Language

1 Introduction

The aims of this chapter are five-fold:

- First, the context of this work is described, which refers to the technologies involved and their evolution, as well as existing major problems;
- Second, the basis of this research work is detailed;
- Third, the main statements of this work are given;
- Fourth, the main contributions of this work are pointed out;
- Fifth, the organization of the document is presented.

1.1 Context

The World Wide Web (WWW or simply Web) as introduced in [Berners-Lee 1989] was first designed as an hypertext information space where humans and machines should be able to communicate [Berners-Lee and Hendler 2001]. Hypertext is human/machine-readable information that contains links to other hypertext and may be accessible through the internet. In order to access/link a specific hypertext document, it is necessary to know its Uniform Resource Identifier (URI), an identifier that uniquely identifies each hypertext in the WWW. Accordingly, the WWW is a network of interlinked hypertext documents represented in HyperText Markup Language (HTML) providing the means to navigate through the information space. The HTML language is a markup language – a language that allows the creation of annotations within a document – for displaying pages in the WWW that may be displayed in a web browser. However, although the content of hypertext documents can be stored in a structured way (e.g. relational databases), the information published in the traditional Web was/is unstructured and therefore is (mostly) consumable by humans only [Berners-Lee, Hendler, and Lassila 2001]. For instance, the hypertext documents usually contain text described in a natural language fashion and discards some other important data (e.g. authorship). In order to tackle this flaw, some syntax was added to hypertext documents through the addition of HTML annotations (e.g. author, title). However, syntax is ambiguous and is primarily meant to be interpreted by humans and not by machines.

As a consequence, searching information in the traditional (syntactic) Web is a task that is mainly performed by humans and therefore it may not be trivial. According to [Morville and Callender 2010], information navigation and search are very hard to formulate, ultimately depending on the user's incomplete, contradictory and evolving requirements, leading to an always incomplete process. Additionally, traditional search engines are not able to properly handle such unstructured hypertext documents because they are syntax based and only allow users to make unstructured queries based on keywords which results in a list of links to other hypertext documents. Accordingly, syntactic search use words or multi-word phrases as search elements (i.e. the keywords). This kind of search engines basically perform two different, but complementary, operations: (i) the search engine crawls (i.e. processes) the documents (e.g. the WWW) and creates an index for the documents based on their content (e.g. page title, page content), (ii) it allows the retrieval of documents based on a set of keywords. Accordingly, to enhance the results, the search engines (may) exploit some Natural Language Processing (NLP) techniques (e.g. tokenization, lemmatization, stemming) to match content previously indexed with the keywords being searched. As a result, the documents containing such content (usually referred to as hits) are retrieved. NLP techniques intend to process data described in a natural language in order to extract relevant information to be used by humans and/or machines. Nevertheless, although using NLP techniques to enhance search, the search is still a syntactic search and presents some of the same flaws. Accordingly, this kind of search usually perform low precision and good recall [Giunchiglia, Kharkevich, and Zaihrayeu 2009]. In this sense, as syntactic search does not take into account the documents' semantics the search results are inherently bad which difficult the task of finding information in the ever growing WWW. For instance, the user usually needs to manually analyze several documents in order to find the one he is interested in. Besides, it does not perform well for answering complex questions as for properly answer such questions it is usually necessary to access and relate information present in multiple documents.

Further on, the great evolution of an ever evolving WWW has highlighted such problems forming some obstacles to its usage. The proliferation of data and information in the WWW led to an increase on the size and complexity of its content, such that the users have problems coping with it causing the information navigation and search processes to be cumbersome and difficult to execute. In order to ease the users' role in these processes, machines should be used. Therefore, it became necessary to evolve from a syntactic traditional Web to a more structured and meaningful Web [Berners-Lee, Hendler, and Lassila 2001] where machines should be able to communicate. This Web is usually referred to as Semantic Web. In this context, the Semantic Web was envisioned as a way of "bringing the Web to its full potential" [Fensel et al. 2003] as an extension of the traditional Web "in which information is given well-defined meaning" [Berners-Lee, Hendler, and Lassila 2001] hence fostering machine-based information processing enabling data sharing across different applications and communities [W3C 2012a]. In this sense, the Semantic Web aims to bridge the gap between a Web of Documents and a Web of Data thus allowing advanced information processing (e.g. reasoning). Accordingly, the information that was originally stored in a structured manner is structurally published in the Web. In this way, the Semantic Web may enable the cooperation between

humans and machines in order to efficiently navigate/exploit through the WWW which ultimately leads to ease the find and process information task. However, in order to fully exploit the Semantic Web it is necessary to connect the data in different sources to create a Web of Linked (Open) Data [Berners-Lee 2009]. The term Linked Data refers to a guide on how to publish data on the Web [Linked Data 2012]. According to this guide, Linked Open Data (or simply LOD) is achievable by connecting different data, information and knowledge through the Semantic Web by using the resources' URIs [Bizer, Heath, and Berners-Lee 2009]. In this sense, querying this new data space where both documents and data are linked opens possibilities not conceivable before. For instance, data from different data sources can be integrated in order to expand the search space thus creating a more complete view. However, querying over the Web of Linked Data also poses new challenges that do not arise in traditional query processing. In this sense, mechanisms supporting information retrieval are required.

Therefore, the evolution of the traditional Web of Documents to a Web of Linked Data (i.e. the Semantic Web) has opened a new set of opportunities to exploit but also some new problems to exceed. For instance, semantic search is now seen as a possibility. Semantic search gives a new meaning to the search term as it perceives the actual meaning of the terms to search and content to retrieve thus providing better results. Accordingly, there are Linked Data search engines that crawl the Web of Data allowing more expressive query capabilities to these systems (e.g. Watson [Watson 2012]). However, when the user aim to find an answer to some kind of question questions (e.g. factoid questions), these systems not always produce the best results. Therefore, mechanisms that help users answering questions are necessary.

Question Answering (QA) is a computer science field that tries to build systems that automatically answer to questions posed by humans in natural language [Kolomiyets and Moens 2011] and it attempts to deal with a wide range of question types (e.g. fact, how, why) [Chali, Joty, and Hasan 2009]. Usually, this kind of systems access structured data in a repository (e.g. knowledge bases) in order to answer the questions. In this sense, the Semantic Web is seen as a tremendous repository of data, where information and knowledge are ready to be exploited by such systems. However, these systems are usually fully automatic and not always perform well. This is mainly because natural language is ambiguous and therefore natural language processing is always a complicated task. In this sense, it is clear that such systems may benefit from the help of humans in such task. For instance, humans may help in the disambiguation process while defining a question. By doing this, humans are bridging the gap between the language that they express themselves and the language that is understandable by machines.

In this sense, the work presented in this document is focused in studying/developing methodologies/processes that may help users to find accurate answers for complex questions which cannot be answered using traditional systems. This includes:

(i) Exceed the users' difficulty to envision the knowledge repository schema;

(ii) Bridge the gap between the natural language expressed by the user and the (formal) language understandable by the repository;

(iii) Process and retrieve relevant information in order to properly answer the user questions.

1.2 Motivation

The work described in this document was developed in the research context of the World Search [World Search 2010] project. Therefore, a brief description of the project is provided as follows:

- First, the World Search project is briefly introduced;
- Second, the observations resulting from the requirement analysis are succinctly described;
- Third, the project's system architecture is presented.

1.2.1 World Search Description

The World Search project [World Search 2010] aims at developing search technologies with semantic relevance for the Portuguese culture and market for both enterprises and public web domains. Accordingly, it aims to provide applications for specific domain that supports domain experts during their quest for information in the Portuguese language. This project should be materialized with the development of two pilots in two well defined domains, namely, the Health Care domain and the Local Public Administration.

In both domains, information is characterized as being stored in an unstructured fashion and without any associated semantics. Thus, finding the necessary information to answer complex questions in such systems is a difficult task. Complex questions, unlike simple questions, usually seek different types of information and merges different answers in order to fully answer the proposed question. So, the quest for information in multiple non-structured repositories is cumbersome and users do not always achieve the best results. Also, although users are aware of the domain, they do not usually know how to access and retrieve information from the repositories. In this sense, it is desirable to add semantics to the information and then to store it in a (semi-)structured manner thus allowing complex question answering through machine-supported processes. By the creation of knowledge bases (i.e. an information repository), one is allowing the development of mechanisms that allows a user driven non-structured search but that might be enhanced with semantic information.

Since the two World Search application domains are too specific and may require knowledge that is not broadly known, within this document context a third application domain is adopted: the open domain knowledge. Hopefully, this domain does not requires any further knowledge by the reader since it is about persons, their roles and activities, places, etc. This domain is

mainly used for exemplifying purposes only. In this sense, Wikipedia [Wikipedia 2012] is seen as a good information space to exploit. Wikipedia is considered as the largest community encyclopedia [Völkel et al. 2006]. Accordingly, Wikipedia users are able to participate in the expansion of the Wikipedia by adding some information to its articles. In this sense, users usually add some information in natural language which ultimately leads Wikipedia to consist primarily of unstructured information. Nevertheless, Wikipedia does not only contain unstructured data but also some structured information, which is captured by the Wikipedia Infoboxes. Wikipedia Infoboxes constitutes a summary of some important information about Wikipedia articles in a structured and easy to read manner. However, Infoboxes are not properly exploited by the Wikipedia search engine neither by the most used search engines (e.g. Google, Bing) [Hahn et al. 2010]. The DBPedia [DBPedia 2007] project is a community effort to extract structured information from Wikipedia – mainly from the Wikipedia Infoboxes but also exploiting some NLP processing upon Wikipedia articles. Accordingly, the DBpedia is a knowledge repository where information from Wikipedia is well described and the different resources are explicitly connected. As a result, the DBPedia knowledge base is considered one of the most important parts of the Linked Data project [Berners-Lee 2008] and therefore it is an appropriate repository to exploit. An example of a common-sense complex question is given in the DBPedia. However, for simplifying purposes, it is only used an excerpt of the original question as depicted in **Sample Question 1**.

Sample Question 1 – All soccer players, who played as goalkeeper for a club that has a stadium with more than 40.000 seats.

Answering complex questions like the one in **Sample Question 1** requires the gathering of data/information from multiple resources. Additionally, these resources are available in multiple and heterogeneous repositories. In the context of this project, a resource is either (i) a text document, (ii) an user annotation of a (part of a) document or (iii) a set of facts in a knowledge base.

1.2.2 World Search Requirements

During the analysis of requirements, the development team observed that the users were interested neither in text-based searches only, nor in formal queries to the structured repository. Instead, users are interested in an elaborated combination of both. I.e. users want to have the chance to make a query that includes free-text and semantic specification of content. This combination is formally captured by the next function:

$$(res', sem\_entities') = query(text, sem\_entities, res)$$

where:

- $text$ is a user entered free-text;
- res is a set of resources (documents, annotations or facts) in which the user is interested for. It serves as example of the resources to retrieve;

- $sem\_entities$ is a (partial) formal specification of the required content based on a model, typically in the form of a set of taxonomy entities or ontological entities. It serves as formal constraints to the query, i.e. only those resources semantically defined/annotated/related with those entities should be retrieved;
- res' is the set of resources retrieved to the user for visualization. This includes both the text-based retrieved resources (documents) and ontology-based retrieved resources (annotations and facts);
- $sem\_entities'$ is the set of relevant semantic entities that (i) belong to the formal modal describing the resources, (ii) is representative of the semantics of the output resources (res') and (iii) enables the user to further refine the query.

Furthermore, three search' related use cases where identified as depicted in Figure 1.

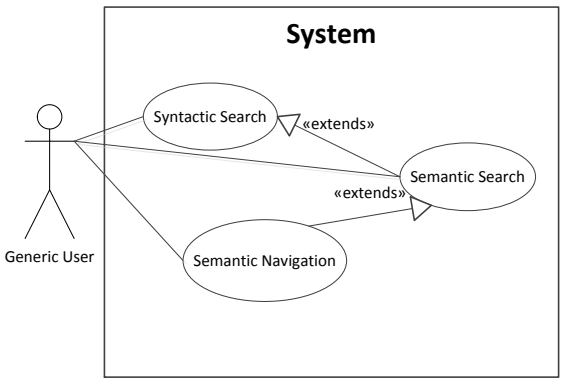

Figure 1 – World Search Use Case Diagram excerpt

Notice that the aims of this thesis work only focus on the semantic search/navigation features of the identified requirements. Accordingly, the syntactic search is not addressed.

1.2.3 World Search Architecture

To fulfill the identified requirements and the overall project purposes, a generic system architecture was defined. This architecture is illustrated in Figure 2.

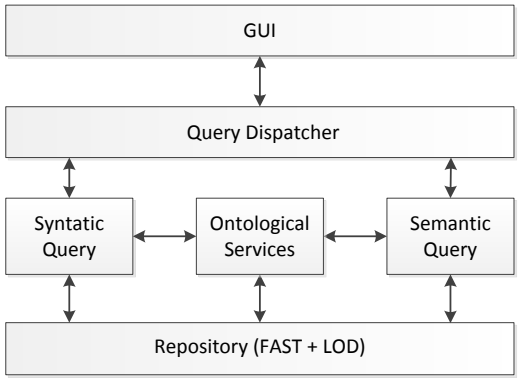

Figure 2 – The architecture of the World Search project

This architecture has six main modules with the following responsibilities:

- Graphical User Interface (GUI) is responsible for the interaction with the user: (i) it enables the user to formulate queries and (ii) presents the respective responses (i.e. the retrieved results);
- Query Dispatcher is responsible for forwarding the queries to the proper answering module;
- Ontological Services module is responsible for managing the semantic information of the system (e.g. maintain the ontologies underlying the system) and providing semantic services (e.g. correspondences between concepts, synonyms) to the other system' modules;
- Syntactic Query is responsible for retrieving resources based on a text-based search only. It might make use of the ontological services to expand the query based on the synonyms of the words specified in the query;
- Semantic Query is responsible for retrieving resources based on the semantic entities specified by the user. The ontological services are exploited in order to perceive the relations between those entities and other ontological entities;
- Repository is where all the data/information supporting both the syntactic and the semantic queries is maintained. Currently, this module is mainly composed by (i) FAST technology [Microsoft-Corporation 2008] for indexing purposes and (ii) a data source meeting the Linking Open Data principles [Berners-Lee 2009].

In the context of this thesis, the focus is on the semantic component of the World Search project. Accordingly, the remaining of the document presents the efforts made in the development of the Semantic Query and the Ontological Services components of the World Search architecture (cf. Figure 2).

Moreover, in the context of this work, it is assumed that the LOD fragment of the Repository component is an ontology-based repository where the data/information/knowledge necessary to answer the user complex questions is maintained. Ontologies may be expressed in a variety of ontology languages (e.g. OWL [Smith, Welty, and McGuinness (eds.) 2004]) that define the ontology entities. Fortunately, most of these languages share the same kind of entities, often with different names but comparable interpretations.

1.3 Thesis Statement

This work advocates that answering complex questions on semi-structured repositories is a task that ultimately depends on the users. Therefore, to foster the task accomplishment users' supporting mechanisms are needed for helping them searching, navigating and querying iteratively, incrementally and interactively such semi-structured repositories in order to bridge the gap between the users' conceptualization underlying the question and the domain conceptualization expressed in the repository that is used to answer such question.

1.4 Research Contributions

In order to achieve the main goal of answering complex questions, several contributions have been proposed. Accordingly, this thesis work describes a set of basic functionalities that helps users in their information navigation and search tasks through ontology described repositories thus allowing the user to answer complex questions. The main contributions of this work are listed as follows:

- The identification of a required set of users' supporting functionalities that a Question Answering system should provide;
- A formal definition of the identified functionalities;
- A prototype containing the proposed functionalities.

Most of these contributions were previously published in [Brandão, Maio, and Silva 2012a].

1.5 Thesis Organization

This document is organized as follows:

- Chapter 1 presents the context of this work while providing its main objectives and motivations. Moreover, it presents the research project where this work is framed. Finally, it presents the thesis statement and enumerates the main contributions of this work.

- Chapter 2 provides a succinct overview over the Semantic Web and some of its associated technologies. Moreover, it describes and defines the ontology modularization process. Finally, it surveys search technologies while relating them to the Semantic Web and presents some state-of-the-art applications.

- Chapter 3 demonstrates some experiments performed with state-of-the-art tools and draws some conclusions according to the observed results.

- Chapter 4 outlines the drafted proposal that supports the complex querying task. Accordingly, the proposed features and architecture are presented. Moreover, the proposed system is formally described.

- Chapter 5 describes the system's implementations details and presents the developed prototype.

- Chapter 6 presents the evaluation performed to the system. Accordingly, it describes the accomplished user study and shows the obtained results. Finally, some conclusions based on the observations are achieved.

- Chapter 7 contains some conclusions about the described work, presents some justifications for the thesis statement, and indicates some future work.

2 Background Knowledge

The purpose of this chapter is three-fold:

- First, a survey over Semantic Web related technologies is presented;
- Second, the Ontology Modularization process and techniques are described;
- Third, an overview over Question Answering and Information Retrieval is presented. Furthermore, a description on state-of-the-art applications is performed.

2.1 Semantic Web Technologies

In order to evolve from the traditional Web to the Semantic Web, some changes had to be made. For instance, for the Semantic Web purpose it is necessary to store information in a structured manner that may be accessed, understood, and manipulated by computers. Hence, it is required to add machine-readable knowledge in the WWW documents. Knowledge representation systems are seen as adequate to gather knowledge as they may be accessed by users and machines [Berners-Lee, Hendler, and Lassila 2001]. In this sense, although machines do not really understand the meanings of the vocabulary, they are able to manipulate the data in a more effective way (e.g. automated reasoning).

In this respect, ontologies are seen as an appropriate formalism to capture and represent knowledge as they provide the vocabulary and its formal specification [Hepp, De Leenheer, and de Moor 2007]. One of the most accepted definitions of ontology is from [Gruber 1993] who defines ontology as “an explicit specification of a conceptualization”. A *conceptualization* refers to the description of the world (the knowledge domain) by the means of concepts and their relationships [Gruber 1995]. *Explicit* means that all the used vocabulary (i.e. concepts and relationships) are explicitly defined [Studer, Benjamins, and Fensel 1998]. Moreover, based on Gruber’s definitions Borst [Borst 1997] defined an ontology as “a formal specification of a shared conceptualization”. Such definition emphasizes the need of agreement between the parties that share the conceptualization. Moreover, *formal* refers to the necessity of such specification be machine readable [Studer, Benjamins, and Fensel 1998].

More recently, Description Logic (DL) has been used to build ontologies. DL is a family of knowledge representation languages that are used to provide “high-level description of the world that can be effectively used to build intelligent applications” [Baader et al. 2007]. *Intelligent* means that the system is able to reason about the explicitly represented knowledge. Therefore, DL provides the means to deal with the knowledge semantics while providing the means to automatic reasoning.

However, at this point the knowledge representation system and the traditional WWW technologies were not compatible. In order to effectively combine the WWW and the knowledge representation systems, knowledge representation and annotation languages had been developed in the top of the WWW infrastructure [van Ossenbruggen, Hardman, and Rutledge 2006]. In this sense, eXtensible Markup Language (XML) and Resource Description Framework (RDF) were proposed. XML is a meta markup language that defines a set of rules in order to encode documents in a way that is both human and machine readable. RDF is a XML-based markup language that provides a simple data model for expressing statements about resources (e.g. hypertext) as (subject-predicate-object) triples. Due to the RDF representation, a new type of databases – the triple stores – where designed in order to store and retrieve triples [Broekstra, Kampman, and Van Harmelen 2002; Bishop et al. 2011]. Moreover, as in traditional databases and database management systems (DBMS), it was also necessary to develop specific mechanisms to access such specific repositories. Despite any RDF model can be serialized in a XML notation, RDF is different from just XML as it has its own data model [Broekstra, Kampman, and Van Harmelen 2002]. Therefore, XML query languages (e.g. XQuery [Boag et al. 2010]) are not suitable to query RDF structures. The SPARQL Protocol and RDF Query Language (SPARQL) [Prud’hommeaux and Seaborne 2008] was developed in order to query RDF data structures and, therefore, to access data stored in these so-called triple stores.

Figure 3 depicts the architecture of the Semantic Web.

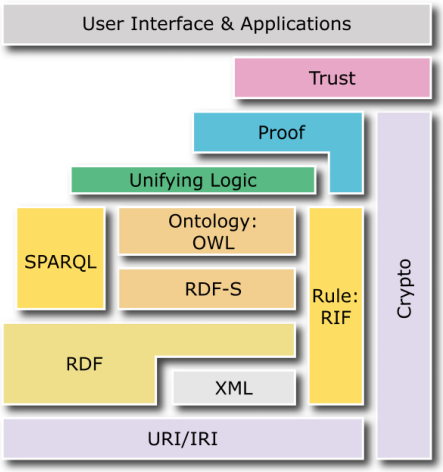

Figure 3 – The Semantic Web Stack [W3C 2007]

Starting from the bottom, the most basic technology in the stack is the URI that identifies each document/resource in the WWW. Further, XML is the adopted structure to serialize the

information while RDF is the language in which information is described. Moreover, in order to add some expressivity to the RDF data model, some other knowledge representation languages for authoring ontologies, providing different (logical) expressivity, had been developed (e.g. Resourse Description Framework Schema (RDF-S) [Lassila and Swick 1999; Brickley and Guha 2000], Web Ontology Language (OWL) [Smith, Welty, and (eds.) 2004]). Further, some Rule mechanisms (an ongoing standard) that allows the description of relations that cannot be described using DL where suggested (e.g. Rule Interchange Format (RIF)[W3C 2012b]). On the top of that is the Unifying Logic layer that is intended to provide a global coherent logical theory. Moreover, the Proof and Trust layers intend to provide explanations on the found answers and measure the belief on the proofs, respectively. Accordingly, the Proof layer intends to provide proofs that an answer in the Semantic Web is correct by (i) proving how it was derived (logic), (ii) checking from which resources it was retrieved, and (iii) checking who provided the data (trust). Moreover, the Trust layer was proposed (i) for verifying if the information come from trusted sources, and (i) for creating different trust policies (policy management) for the Semantic Web. Further, Cryptography layer is intended to add some security to the Semantic Web by ensuring and verifying that information is produced from trusted sources. Finally, User Interface & Applications is the final layer that enables users to interact with the semantic web applications.

2.2 Ontology Modularization

With the growing of the Semantic Web the usage of ontologies has become predominant. However, despite ontologies are used to describe a specific domain of interest, their size and complexity tends to increase too [Del Vescovo et al. 2011]. This is emphasized by the fact that ontologies are typically developed as monolithic blocks of information, i.e. all the (domain) knowledge is gathered into a single ontology (e.g. OWL document). Thus, ontology understandability decreases as its complexity increases which consequently leads to an increase in human effort in apprehend and reuse them [Stuckenschmidt and Schlicht 2009]. Hence, current tools do not effectively support the navigation through ontologies [Dzbor et al. 2006], especially those inexperienced and non-experts users. In this sense, ontology modularization is a recent research field that deals with the problem of oversized and complex ontologies in order to enable its usage.

Ontology Modularization is typically seen as an engineering process characterized by addressing the problem of creating modules from existing ontologies [Stuckenschmidt and Klein 2004]. In this sense, modularization refers to a situation where a thing (e.g. an ontology) exists as a whole but can also be seen as a set of parts (the modules) [Parent and Spaccapietra 2009]. By splitting an ontology into smaller parts, one is allowing the selective use of knowledge which (i) facilitates ontology reusability and share-ability [Stuckenschmidt and Schlicht 2009], (ii) reduces the human effort in understanding such ontology [Parent and Spaccapietra 2009], (iii) empowering the ontology manipulation, maintenance and evolution tasks [Bezerra et al. 2009] and (iv) improves the usage of reasoners (e.g. by Distributed

Reasoning, by incremental reasoning) [Del Vescovo et al. 2011]. Yet, another advantage of ontology modularization is the possibility of knowledge contextualization (different parts of the ontology may correspond to different contexts) and knowledge personalization, i.e. ownership and authorization [Parent and Spaccapietra 2009].

Although there are several modularization techniques, some have gained more relevance than others. Moreover, each modularization technique usually varies in terms of requirements and goals [Parent and Spaccapietra 2009]. In the context of the World Search project, it was envisaged the adoption and combination of three main modularization techniques [Parent and Spaccapietra 2009]: (i) ontology partitioning, (ii) module extraction and (iii) ontology summarization. Considering that these techniques are further exploited in sections 4.2.6 and 5.6, they are described and formally defined next.

2.2.1 Ontology Partitioning

For the proper understanding of the rest of this document a formal definition about ontology is provided. Accordingly, an ontology can be minimally defined as follows.

Definition 1 (Ontology) – An ontology $\mathcal{O}$ (also known as knowledge base) is a tuple $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is the terminological axioms and $\mathcal{A}$ is the assertional axioms. Both are defined based on a structured vocabulary $\mathcal{V} = (\mathcal{C}, \mathcal{R})$ comprised of concepts (or classes) $\mathcal{C}$ and roles (or properties) $\mathcal{R}$. Concepts (and roles) axioms are of the form $C \sqsubseteq D$ ($R \sqsubseteq S$) or $C \equiv D$ ($R \equiv S$) such that $C, D \in \mathcal{C}$ ($R, S \in \mathcal{R}$) respectively. For a set of individuals $\mathcal{I}$, concepts and roles assertions are of form $C(a)$ or $R(b, c)$ such that $C \in \mathcal{C}$, $R \in \mathcal{R}$ and $a, b, c \in \mathcal{I}$.

Yet, it is worth mentioning that ontology entities are not necessarily named. In fact, ontology entities can be constructed out of other entities. As an example, a concept may be created out of a restriction of a role. Moreover, ontology languages are often extended by entity languages adding operators (e.g. concatenation of strings) to manipulate the ontology entities.

The semantics related to an ontology is provided by an interpretation $\mathfrak{I}$ over a domain Δ such that it maps: (i) the elements of the domain to the ontology instances, (ii) the subsets of the domain to the ontology concepts, and (iii) the binary relations on the domain to the ontology roles.

Finally, information on ontology-based repositories is typically accessible through a query language such as SPARQL.

An ontology partitioning technique identifies the key topics of an ontology and splits it into several (probably disjoint) fragments [Stuckenschmidt and Schlicht 2009; d' Aquin et al. 2009]. Typically, each key topic gives rise to a fragment which is usually called as module (cf. **Definition 2**). Accordingly, ontology partitioning might (i) help understanding the ontology by representing its modular structure [Del Vescovo et al. 2011], (ii) help ontology visualization by splitting the ontology into visualizable modules and by identifying key topics on the ontology,

and (iii) enhance the usage of reasoning tools [Stuckenschmidt and Schlicht 2009]. Finally, it is easier to maintain large ontologies if they are decomposed into several (smaller) modules [Stuckenschmidt and Schlicht 2009].

Definition 2 (Module) – A module $\mathcal{M}$ of an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is defined as $\mathcal{M}(\mathcal{O}) = (\mathcal{T}', \mathcal{A}')$, where $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{A}' \subseteq \mathcal{A}$ are the axioms dealing with (i) concepts $\mathcal{C}'$, (ii) roles $\mathcal{R}'$ and (iii) individuals $\mathcal{I}'$ such that: (a) $\mathcal{C}' \subseteq \mathcal{C}$, (b) $\mathcal{R}' \subseteq \mathcal{R}$ and (c) $\mathcal{I}' \subseteq \mathcal{I}$ respectively. Accordingly, an ontology module is *per se* an ontology too.

A partition technique is formalized as follows.

Definition 3 (Ontology Partitioning) – The Partitioning task is seen as a function $\rho: \mathcal{O} \rightarrow \mathcal{P}$ where an ontology $\mathcal{O}$ is splitted into a set of modules $\mathcal{P}$ with N elements (modules) such that $\mathcal{P} = \{O_1, O_2, \dots, O_N\}$.

2.2.2 Module Extraction

A module extraction technique aims to extract a focused fragment (or module) of the original ontology given a specific topic of interest [Hussain and Abidi 2010]. The topic of interest is captured by the notion of signature (cf. **Definition 4**). This way, a contextualized fragment may be extracted in order to reduce the overall ontology's complexity and therefore allowing its selective usage (e.g. for reuse and/or sharing purposes) [d' Aquin et al. 2009].

Definition 4 (Signature) – A signature $\mathcal{S}$ to extract a module $\mathcal{M} = (\mathcal{T}', \mathcal{A}')$ from $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is defined as $\mathcal{S}(\mathcal{O}) = (\mathcal{T}'', \mathcal{A}'')$ where $\mathcal{T}'' \subseteq \mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{A}'' \subseteq \mathcal{A}' \subseteq \mathcal{A}$ are the axioms (concepts $\mathcal{C}''$, roles $\mathcal{R}''$ and individuals $\mathcal{I}''$) specifying the context of the module to be extracted such that: $\mathcal{C}'' \subseteq \mathcal{C}' \subseteq \mathcal{C}$, $\mathcal{R}'' \subseteq \mathcal{R}' \subseteq \mathcal{R}$ and $\mathcal{I}'' \subseteq \mathcal{I}' \subseteq \mathcal{I}$.

A module extraction technique is formalized as follows.

Definition 5 (Module Extraction) – The Module Extraction task is seen as a function $\sigma: (\mathcal{O}, \mathcal{S}) \rightarrow \mathcal{M}$ where an ontology module $\mathcal{M}$ is extracted from an ontology $\mathcal{O}$ according to a given signature $\mathcal{S}$.

2.2.3 Ontology Summarization

An ontology summarization technique provides a succinct representation (or compressed version) of the ontology (referred to as summary) emphasizing the topics contained in an ontology according to visualization and navigation purposes [Zhang et al. 2009; Li, Motta, and d' Aquin 2010]. Accordingly, ontology summarization may help users perceiving the overall schema/ontology by reducing its size and/or complexity to ones perceptible by the user [Li, Motta, and d' Aquin 2010]. Thus, it may be benefit the GUI by serving visualization and navigation purposes.

Definition 6 (Summary) – A summary description $\mathcal{D}$ of an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is defined as $\mathcal{D}(\mathcal{O}) = (\mathcal{T}', \mathcal{A}')$ where $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{A}' \subseteq \mathcal{A}$ are the axioms specifying the concepts $\mathcal{C}'$, the roles $\mathcal{R}'$ and the individuals $\mathcal{I}'$ that summarize the ontology such that: $\mathcal{C}' \subseteq \mathcal{C}$, $\mathcal{R}' \subseteq \mathcal{R}$ and $\mathcal{I}' \subseteq \mathcal{I}$ respectively.

Ontology summarization technique is then formalized.

Definition 7 (Ontology Summarization) – The Ontology Summarization task is seen as a function $\varphi: \mathcal{O} \rightarrow \mathcal{D}$ where a description $\mathcal{D}$ is generated to summarize the ontology $\mathcal{O}$.

It is worth notice that from the perspective of an ontology, the notions of (i) module ($\mathcal{M}$), (ii) signature ($\mathcal{S}$) and (iii) summary ($\mathcal{D}$) have similar formal definitions. However, these notions differ on their purpose and in extension (in terms of set inclusion), such that:

- $\mathcal{S} \subseteq \mathcal{M} \subseteq \mathcal{O}$
- $\mathcal{D} \subseteq \mathcal{M} \subseteq \mathcal{O}$

No relation can be defined between $\mathcal{S}$ and $\mathcal{D}$.

2.3 Question Answering

The purpose of this section is two-fold:

- First, an overview over Question Answering and Information Retrieval is presented;
- Second, a survey is performed upon current state-of-the-art search tools.

2.3.1 Overview

Lots of attention has been given to the QA field in order to improve Information Retrieval (IR) in computerized systems [Kolomiyets and Moens 2011]. Current QA systems are already prepared to answer simple questions (e.g. Wolfram Alpha [Wolfram Alpha 2012]). However, these systems do not perform so well while answering complex questions even more if they try to answer open questions. Open-domain question answering tries to answer about anything in the world and usually seeks information in large data spaces (e.g. DBPedia). Therefore, making complex questions against current systems is often a complicated task and occasionally some questions are even impossible to answer although the needed information is available. In this sense, QA systems are being improved for answering not only simple questions but also some more complex questions.

In this context, the evolution of the traditional Web to a Semantic Web also allowed the evolution of the search paradigm as well. Accordingly, a new search paradigm emerged – the Semantic Search. Semantic search tries to improve search precision by analyzing the semantic knowledge representation of the documents and the query to retrieve the semantically relevant documents to the query at hand. In other words, the semantic search exploits the

meaning of the words, instead of their syntactic representation [Giunchiglia, Kharkevich, and Zaihrayeu 2009].

Now, due to the evolution of the WWW, open information repositories described by means of ontologies are becoming very common both in the web (e.g. DBPedia [DBPedia 2007]) and enterprises (e.g. World Search [World Search 2010]). These repositories are often referred as Linked-Open Data repositories, or simply LOD. Due to its well-formed structure (and sometimes semantics), the semantic search paradigm gains more relevance and a new set of applications and demands arise. However, these new kind of repositories have some other problems. For instance, (i) the schemas of these repositories are very dynamic, and (ii) the structural and semantic requirements putted upon the schemas and upon the data are heterogeneous, dynamic and potentially unlimited, hence rising operational issues. Moreover, the user does not always know the vocabulary and/or the schema that (s)he is trying to query. Accordingly, both information provider and consumer usually have different conceptualizations and/or perspectives of the domain of interest, adding yet another (semantic) gap between the provider and the consumer.

Therefore, supporting tools are needed in order to fully exploit this kind of repositories, such that the questions are written in a user-demand basis, and not made available by the developer. In particular, the complex query building task is intended to be opaque for the users, as they often lack fundamental required skills to perform such task: (i) to know the underlying schemes/ontologies and (ii) to specify the queries formally (e.g. by means of query language). For instance, although SPARQL is typically used to query open data repositories, most users are not able to write SPARQL queries. Moreover, the complexity of the data in the repositories and their underlying ontologies may overwhelm the users which highlight the needs for machine-supporting mechanisms for navigating through such data.

Accordingly, different proposals have been presented [Goker and Davies 2009] in order to mitigate the users' information navigation and search tasks that will ultimately lead them to properly create their questions. In the context of this work, it is important to highlight two different approaches: (i) Faceted Search [Hearst 2006; Tunkelang 2009] and (ii) Ontology-supported navigation.

The former is becoming a predominant approach as it allows the consumer to drive the search in a structured way. Faceted Search relies on the attributes of the elements in a collection (the facets) [Hearst 2006] to create a classification system along multiple dimensions. For example, a collection of cars might be classified using a brand facet, a fuel facet, etc. Moreover, while accessing structured repositories described by ontologies, Faceted Search might exploit these ontologies for creating the facets. Yet, the main advantage of Faceted Search is that it is also able to exploit non-structured data to create the facets (e.g. using NLP techniques for entity extraction). However, it has some drawbacks especially evident when dealing with fairly large schemas [Teevan, Dumais, and Gutt 2008], mainly because the task is performed by the user with minimal machine support. For instance, maintaining the focus of the search is not so trivial for the user. Moreover, Faceted Search does not properly express more complex

relationships (i.e. real world relationships). Additionally, taxonomies have revealed to be insufficient to navigate as they do not always capture the user's perspective [Warren 2006].

Ontology-supported navigation is a recent research field seen as essential to make sense of ontologies contents and organization [Franconi, Guagliardo, and Trevisan 2010; Motta et al. 2011]. It aims to assist the user in comprehending, searching and retrieving information from repositories described through ontologies.

Summarizing, this new Web of Linked Data has opened new possibilities to the QA field. Accordingly, QA systems might be improved on their tasks by adding semantics to the data they exploit. However, in order to fully exploit these systems, users should also be a relevant part in the search process. In this sense, machine-based mechanisms supporting/guiding the user in her/his role are fundamental (e.g. Faceted Search).

As the main focus of this work is complex question answering through repositories described by ontologies such as LOD repositories, it is desirable to concentrate on the systems that allow the exploitation of such repositories. In this sense, next sub-sections briefly describe applications that query the DBPedia repository in order to help users finding information in a structured fashion.

2.3.2 Faceted Wikipedia Search

Faceted Wikipedia Search [Hahn et al. 2010] is an alternative search engine that allows users to ask complex question against the Wikipedia knowledge base. Accordingly, it exploits the data on the DBPedia ontology/repository in order to allow users to query Wikipedia like a structured database. Moreover, in order to help inexperienced users, Faceted Wikipedia Search relies on the faceted search paradigm. Accordingly, faceted search allows users to ask questions to the repository in an exploratory fashion. Moreover, it allows users to easily navigate through the information space by combining text search with filtering techniques. In this sense, facets allow the narrowing of the information space along multiple dimensions (cf. Figure 4).

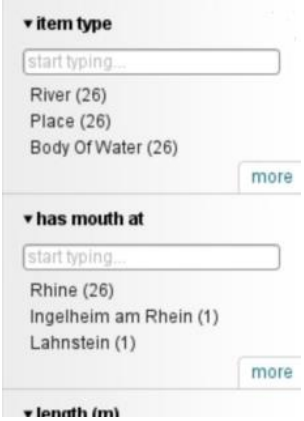

Figure 4 – Sample facets in the Faceted Wikipedia Search engine

Further, by selecting multiple facets the user is constructing the desired query. For instance, the user might ask the system a complex question like “Which are the Rivers that mouth in Rhine and are longer than 50000 meters” as depicted in Figure 5.

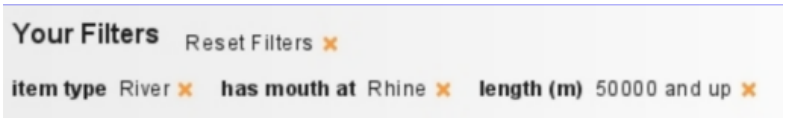

Figure 5 – Complex question formulation using facets

Faceted Wikipedia Search relies on two components to achieve its purpose:

(i) DBPedia Information Extraction Framework, a tool for extracting structured knowledge from the DBPedia (i.e. it queries the DBPedia repository). Accordingly, this is the component used to answer the questions;

(ii) Neofine search, a commercial search engine that implements the faceted search paradigm. Based on the query results, this component generates the facets to display.

In this sense, Faceted Wikipedia Search eases the users’ task of complex querying by allowing users to ask complex queries against Wikipedia based on the schema/ontology and information extracted from many different Wikipedia articles. Moreover, Faceted Wikipedia Search uses some heuristics to determine which facets are more relevant for the user. However, Faceted Wikipedia Search fails to help user finding relations between different concepts/properties, thus undermining the users’ quest for information. Moreover, it does not allow the user to restrict the returned values by adding different filters to different entities (e.g. *Which Portuguese soccer players played in an Italian team located in Milan?*).

2.3.3 RelFinder

In [Heim, Lohmann, and Stegemann 2010] the authors present an approach for interactively discover relationships via the Semantic Web thus emphasizing the human aspect on the relationship discovery. Accordingly, authors try to efficiently exploit the relationships over knowledge domains by taking advantage of the Semantic Web and by adding the user in the search process. Such approach is concretized as stated by the ORVI process, which is a general process for finding relationships. ORVI is defined as a four step process (cf. Figure 6):

(i) Object Mapping is when the user starts his quest. Accordingly, the system performs the mapping between the user input and a unique object in the repository. This mapping is usually automatic and therefore it does not require manual disambiguation. However, if manual disambiguation is required, the system should support the user (e.g. auto-completion);

(ii) Relationship Search occurs after all the selected entities are mapped to their unique values. Accordingly, is when the system finds all the relationships between the previously identified objects (i.e. entities);

(iii) Visualization is responsible for properly presenting the found relationships to the user. Accordingly, some mechanisms for facilitate users' understanding should be applied;

(iv) Interactive Exploration is the last step of the process and it is when the user explores the found relationships. Once again, the system should support the users while they try to find the relevant relationships (e.g. highlighting).

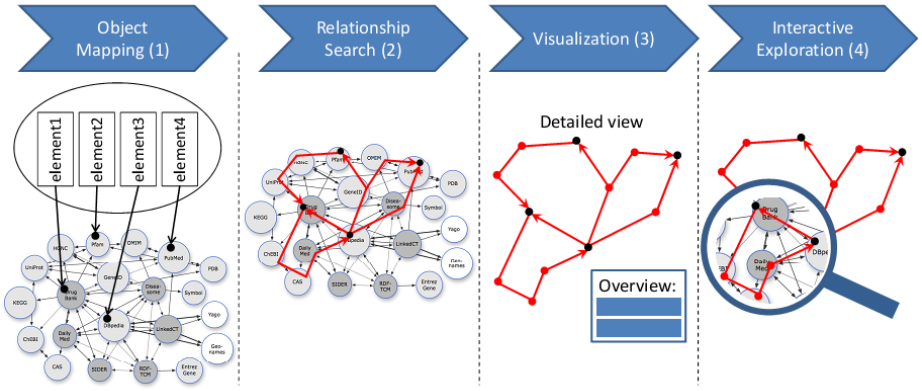

Figure 6 – The ORVI process [Heim, Lohmann, and Stegemann 2010]

The OVRI process has been implemented in RelFinder [RelFinder 2012], an online user-centered tool for interactively discover relationships over knowledge bases. Accordingly, RelFinder generates SPARQL queries on the client side that can be sent to any SPARQL endpoint. The main drawback of this tool is that its' only objective is to find relationships between elements of the repositories. Therefore, this tool does not aim to answer complex questions. However, the relationship discovery is an important feature in the users' information quest. For instance, this feature could enhance applications like the Faceted Wikipedia Search. Moreover, this tool/process lacks of the ability to find the related entities of a given element (e.g. Which elements are linked to the "Person" element?).

2.3.4 gFaceted

In [Heim, Ziegler, and Lohmann 2008; Heim and Ziegler 2011] the authors propose a method that allows/supports users to efficiently access and explore large amounts of data over the Semantic Web. In this sense, it is proposed Facet Graphs, a new approach to help users formulating search queries against semantic repositories. To achieve such goal, this method (i) allows the narrowing down large sets of data based on the concept of faceted search, and (ii) exploits the graph-based structure of the Semantic Web. Accordingly, facets are represented as nodes of a graph that can be iteratively and interactively refined based on the users' requirements. In this sense, the semantic data is seen as a graph that provides a coherent representation of the (search) facets thus highlighting the relationships between such facets.

The proposal relies on a three-stage model consisting of:

(i) Goal formation – the initial stage where the user typically specifies one or more concepts to determine the initial search space;

(ii) Construction of the exploration space – the stage where the user incrementally constructs the search space by exploiting the relationships between the starting concepts and others linked to them;

(iii) Multi-perspective exploration and sense-making stage – the final stage where the user explores the space and "make sense of the information and relations contained in it". In this sense, the user may choose to finish the search if s/he is satisfied with the results or to refine/rebuild the search space by persecuting to the second stage of this model again thus highlighting the iterative fashion of the proposal.

The main advantages of such approach are (i) the prevention of over-cluttered graphs by using facets to group instances, (ii) the explicit representation of relations between facets, and (iii) the representation of a single coherent visualization. In this sense, it is argued that users benefit from this approach as it prevents them from losing track of the relationships over the different facets.

The three-stage model has been implemented in the gFacet [gFacet 2012] tool, an online prototype that queries repositories through SPARQL and visually represents the facets in a graph as depicted in Figure 7.

Figure 7 – Faceted graph navigation with the gFacet tool

However, similarly to Faceted Wikipedia Search, this tool does not allow the intersection of multiple filters (e.g. *Which Portuguese soccer players played in an Italian team located in Milan?*). Moreover, the user understandability decreases when dealing with vast number of facets. Finally, faceted search has some limitations when dealing with vast number of entities.

2.3.5 Virtuoso SPARQL Query Editor

SPARQL Endpoint is a SPARQL protocol (web) service that enables humans/machines to query a knowledge base through the SPARQL language. However, a SPARQL Endpoint is mostly conceived to be machine-friendly rather than human-friendly. In this sense, the SPARQL Endpoint front-ends usually do not provide any help while composing the SPARQL query (e.g. syntax hints, information about the repository to query). Moreover, the returned results are meant to be processed by machines and therefore are not easily handled by humans.

The Virtuoso SPARQL Query Editor [Virtuoso SPARQL Query Editor 2012] is a SPARQL Endpoint used to query the DBPedia knowledge base. As most SPARQL Endpoints, this tool does not support the user in the information quest and it requires the user to know (i) the DBPedia' schema/ontology, and (ii) the SPARQL language syntax. Unfortunately, most users are not aware of the repository' ontology and are not able to construct SPARQL queries. Further, the query results are presented in simple tables which, in some cases (e.g. very complex queries), may be difficult for users to read.

3 State-of-the-Art Assessment

This chapter assesses the state-of-the-art search tools previously described. As that, a trial was conducted to investigate the following:

(i) The ways users make use of these repositories when trying to respond to complex questions;

(ii) How users expect to be supported on the question answering task.

The remainder of this chapter is organized as follows:

- First, the trial set-up is presented;
- Second, the results obtained are described;
- Third, a discussion about the obtained results is presented;
- Fourth, some conclusions on state-of-the-art tools are taken.

3.1 Set-Up

This section describes the carried out trial as follows:

- First, the questions of the experiment are defined;
- Second, the study participants and the used tools are described;
- Third, the process in which the experiment was conducted is shown.

3.1.1 Questions

The user study consisted in responding four different (complex) questions (cf. Table 1). The difficulty level of the question increases according to the number of the question – the first question is the easiest one and the fourth the harder. For instance, the first question may be answered by accessing a single web page. Accordingly, the difficult consists in finding the web page that contains the information needed and then get the wanted answer in its content. In

contrast, to answer the remainder questions, several quests for information in the Wikipedia/DBPedia repository are required. Accordingly, the necessary information to answer such questions (from question two to four) are not explicitly stated in any document or fact. Instead, it is necessary to combine several semantic entities, documents and facts to get the appropriate answer. Accordingly, the complex questions' answer typically consisted in a set of resources. Finally, it was ensured that the given questions could be answered using the proposed tools.

The overall time limit for answering all the questions was 30 minutes. Despite that, users were suggested to answer the questions within 3, 7, 10 and 10 minutes, respectively (as depicted in Table 1).

Table 1 – Questions description

No.	Question	Time
1	Which is the total population of Las Vegas (United States of America city)?	3
2	Name twenty (20) cities with a population over ten million (10 000 000) inhabitants.	7
3	Name fifteen (15) persons whose birth place and death place was Chicago (United States of America city).	10
4	Name ten (10) persons whose birth place and death place was Chicago (United States of America city) and that were born after 1st January of 1900.	10

In the end, the users were asked to answer a questionnaire. This questionnaire served to evaluate the users' feedback on the *usefulness,* and *usability* of the proposal but also their *intention to use* such approach [Ong et al. 2008]. In this sense, *usefulness* is intended to measure "the degree to which a person believes that using a particular system would enhance his or her job performance". Moreover, *usability* measures the ease with which the user uses the tool. *Intention to use* measures the users' intent to use the application to perform their tasks.

3.1.2 Participants and Tools

The study participants were asked to answers the questions using the Wikipedia [Wikipedia 2012] traditional search, RelFinder [RelFinder 2012], gFacet [gFacet 2012], Virtuoso SPARQL Query Editor [Virtuoso SPARQL Query Editor 2012], and the DBPedia pages (cf. section 2.3). At the trial time the Faceted Wikipedia Search [Faceted Wikipedia Search 2012] tool was not available.

Moreover, the users were advised to disregard previous knowledge as much as possible. Consequently, the users were only allowed to answer questions with the results obtained with the available applications. Moreover, the users were asked to answer the questions individually.

Additionally, the gFacet tool was tweaked to access the DBPedia SPARQL Endpoint. The other tools were already accessing the Wikipedia or the DBPedia databases.

Further, the experiments occurred in a lab so that the team could witness the efforts and the adopted approaches. No further restrictions were made.

3.1.3 Procedure

Five users were selected and characterized according to their proficiency about the available technologies (e.g. SPARQL). The users were all informatics students and therefore very familiar with Information Technology (IT) technologies. Accordingly, the 5 users were very familiar with the Wikipedia repository and its traditional search engine (or any general purpose search engine) and faceted search. Moreover, 2 users were unaware about the DBPedia technology and, consequently, unaware about its schema/ontology. Further, these same users were also unaware about the Semantic Web technologies, such as ontologies or SPARQL. The other three users were familiar with the DBPedia repository and its underlying ontology/schema. Further, these users were also familiar with the SPARQL language. Finally, users have never used the tool RelFinder or gFacet. Table 2 depicts the users' characterization. Notice that the proficiency is measured in a numeric value from one (1) to five (5) where 1 means *not familiar* and 5 means *very familiar*.

Table 2 – Description of the test subjects

	Proficiency level			**Gender**	**Academic Qualifications**
	Wikipedia	**DBPedia ontology/schema**	**SPARQL**		
User 1	5	4	4	Female	Licentiate
User 2	5	3	4	Male	Master's degree
User 3	5	2	3	Male	Licentiate
User 4	5	1	1	Male	Licentiate
User 5	5	1	1	Male	Licentiate

The study began with a short introduction and explanation of the trial and the corresponding technologies. Then, a short presentation and demo of the applications was performed. Afterwards, users were asked to answer the proposed questions and questionnaire.

3.2 Results

This section demonstrates the obtained results in the experiment as follows:

- First, the answers to the questions in the experiment are detailed;
- Second, the answers on the questionnaire are presented.

3.2.1 Answers

The users' success on the proposed exercise was evaluated in two different ways:

(i) A Boolean evaluation (true or false) on if the users were able or not to answer a question;

(ii) The evaluation, in percentile, of the correctness of the given answers (i.e. measure the quality of the answers).

The results of the evaluation are depicted in Table 3, which demonstrates the analysis on the answered (✓) and not answered (×) questions.

Table 3 – Analysis on the answered questions.

	Question 1	Question 2	Question 3	Question 4
User 1	✓	✓	✓	✓
User 2	✓	✓	✓	✓
User 3	✓	×	×	×
User 4	✓	×	×	×
User 5	✓	✓	✓	×

Figure 8 depicts the percentile of users that answered the questions based on data from Table 3.

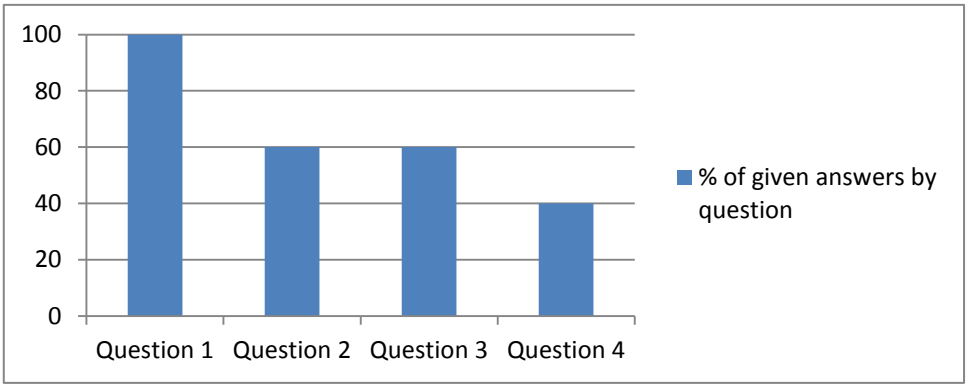

Figure 8 – Percentile of given answers grouped by question

Moreover, the second evaluation was achieved by measuring the correctness of the answers as depicted in Table 4. According to this scrutiny, 3 users correctly answered the first question while 2 users gave a wrong answer. Hence, all the users found the necessary resources to answer this question. The second question was answered by three users. However, their answers were incomplete – one user indicated 16 of the 20 asked resources while the other indicated 17. Moreover, the other user only encountered 1 resource. The third question was correctly answered by 2 users. Moreover, 1 user indicated 2 resources of 10. Finally, only 2 users gave an answer to the fourth question and that is 100% correct.

Table 4 – Percentile of correct resources retrieved in each question.

	Question 1	Question 2	Question 3	Question 4
User 1	100%	75%	100%	100%
User 2	100%	80%	100%	100%
User 3	0%	0%	0%	0%
User 4	100%	0%	0%	0%
User 5	0%	5%	20%	0%

Figure 9 depicts the percentile of totally correct answers given for each question.

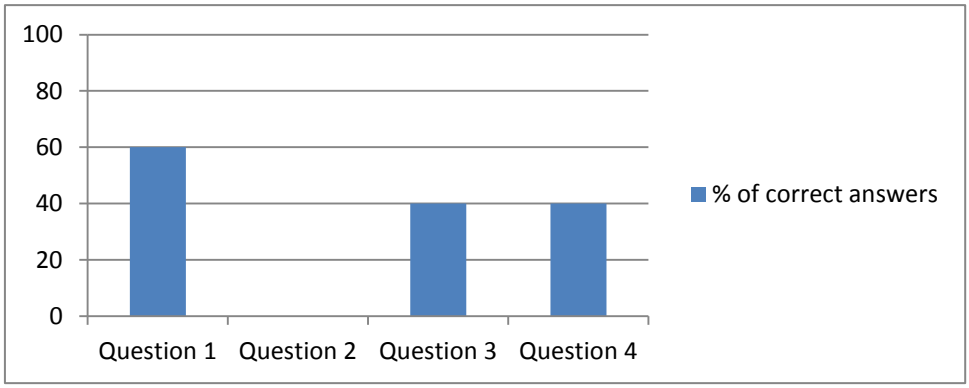

Figure 9 – Percentile of correct answers given (grouped by question)

Moreover, Figure 10 depicts the overall correctness of the answers grouped by question. This is calculated through the average of the correctness of each answer to each question.

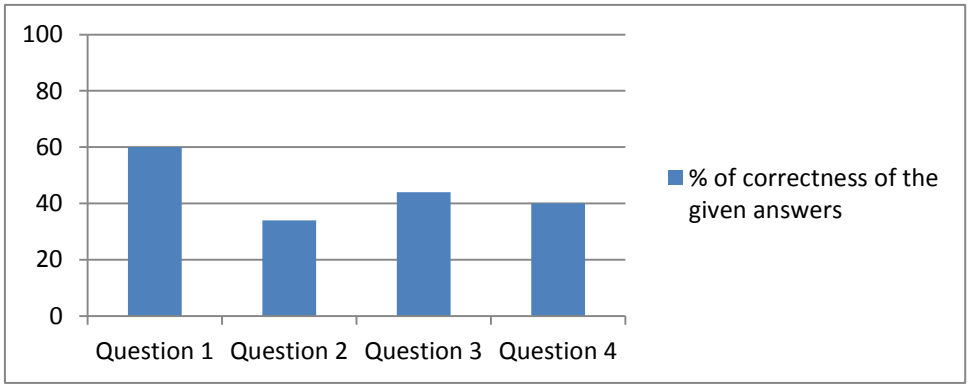

Figure 10 – Correctness of the overall answers (grouped by question)

Additionally, the efficiency of the tool(s) used to answer each question was measured. In this sense, according to the given answers, it was evaluated (i) if the tool(s) used allowed the user to given any answer, and (ii) the tools efficiency (when applicable) for answering the different questions. Table 5 illustrates the tools used by the users for answering each question.

Table 5 – Tool(s) that the users used to answer each question.

	Question 1	Question 2	Question 3	Question 4
User 1	SPARQL	SPARQL	SPARQL	SPARQL
User 2	Wikipedia search	SPARQL	SPARQL	SPARQL
User 3	Wikipedia search	RelFinder+gFacet	RelFinder+gFacet	RelFinder+gFacet
User 4	Wikipedia search	RelFinder+gFacet	RelFinder+gFacet	RelFinder+gFacet
User 5	Wikipedia search	Wikipedia search	Wikipedia search	Wikipedia search

Figure 11 also gives a perspective on which tools were used for answering each question. By the analysis of Figure 11, it is visible that the users opted for using the Wikipedia traditional search for answering simple questions. Yet, most proficient SPARQL users elected the Virtuoso SPARQL Query Editor as the main tool for answering complex questions. Moreover, as expected, non-proficient users in SPARQL and on the DBPedia schema tried to answer the proposed questions by using Wikipedia search or a combination of both RelFinder and gFacet.

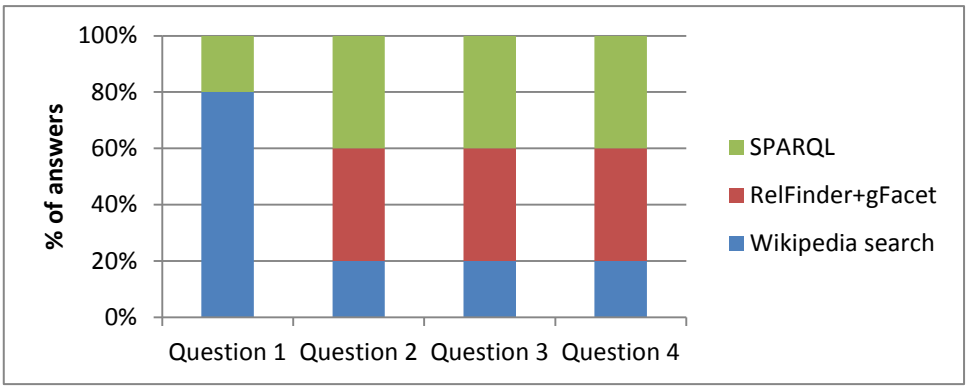

Figure 11 – Percentile of used tools for answering each question

Moreover, the tools efficiency is inferred by a correlation between the correctness of the answers provided by the user with the tools they used to obtain such answers (cf. Figure 12). Notice that in the first question the combination of the RelFinder and the gFacet tools was not considered because none of the users answered the question using these tools. Accordingly, it is clear that the combination of the RelFinder and the gFacet tools are not used for answering simple questions. It also shows that this combination does not perform well for answering complex questions. Further, the results shows that these tools not even allow to give any answer. On the other hand, Wikipedia search revealed to produce good results when answering simple questions. However, the answers given through the Wikipedia search may contain errors. This is mainly because the user needs to find the answer by himself (as occurred in Question 1 with users 3 and 5). Finally, the Virtuoso SPARQL Query Editor is the tool that allowed the users to more successfully respond to the simple and/or complex questions.

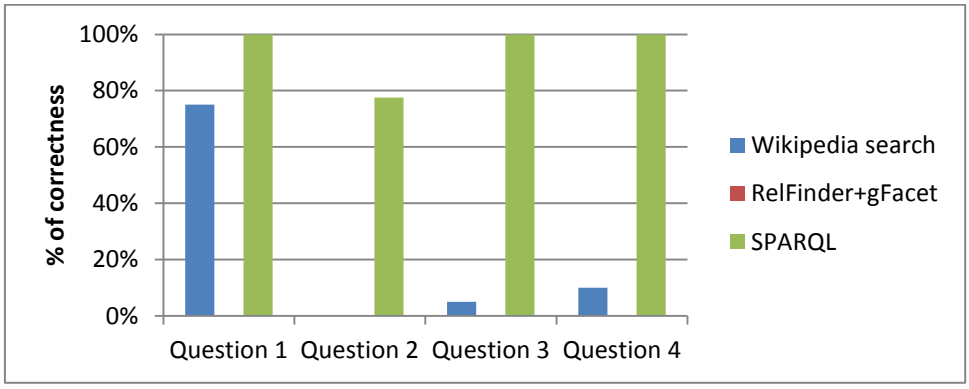

Figure 12 – Correctness of the given answers according to the used tools

Finally, Table 17 depicts the time that users needed to answer all the questions. According to it, only two users finished answering all the questions at approximately the end of the time (i.e. 30 min). However, those users did not fully answer the second question of the exercise.

Table 6 – Time used to answer all the questions.

User	Time (min)
User 1	≈30
User 2	≈30
User 3	N/A
User 4	N/A
User 5	N/A

3.2.2 Questionnaire

The results obtained from the questionnaire are described next.

The first question of the questionnaire asked the users if they were happy with the proposed tools and if they improved their performance while answering complex questions in comparison with any other systems. While answering this question, proficient SPARQL users stated that the Virtuoso SPARQL Query Editor was indeed a good tool for answering complex questions. However, non-proficient SPARQL users said that these tools were inefficient and did not helped them answering the proposed questions.

Further, the users were asked to say if they found the tools easy to use. Most users said that not all the tools were straightforward. For example, it was said that the gFacet tool functioning was not transparent and it was not clear its usage. Moreover, SPARQL proficient users that already knew the DBPedia ontology/schema stated that Virtuoso SPARQL Query Editor was easy to use. Moreover, it was stated that the RelFinder tool, although it is easy to use, it did not accomplished its goals properly. This is mainly because it did not show relationships that actually existed in the repository' underlying schema.

Furthermore, the questionnaire questioned the users about the encountered problems while answering the questions. While some users pointed that it was required (or at least preferential) to know the SPARQL language and the DBPedia ontology/schema to fully answer the questions, others stated that the search methodology of the tools was not clear. Moreover, some users were concerned about the different DBPedia namespaces that could lead to some problems while creating SPARQL queries.

Further on, most of the users stated that they were not fully satisfied with the returned results. Accordingly, RelFinder and gFacet did not produce any (good) results. However, users indicated that they were satisfied with the Wikipedia Search while answering simple questions. In this sense, some users stated that they would use Wikipedia Search for answering such questions. Moreover, proficient users stated that they would only use Virtuoso SPARQL Query Editor for answering complex questions. However, they all agreed that RelFinder and gFacet should not be used to perform such tasks.

Moreover, when asked about any other tool they know for question answering only one user pointed out a different tool, namely the Yago2 spotlx tool [Yago2 spotlx 2012].

Finally, the users stated that it may be useful a function that shows all the relationships starting from a given entity and which of those occur more often (e.g. ranking). Further, users suggested that the tools functioning should be more transparent and should provide more control to the user. Accordingly, the search methodologies should be more intuitive. Moreover, the tools performance should be enhanced (e.g. the RelFinder and the gFacet tools take long time to return results). Finally, users stated that the user interface could be enriched with some semantic information.

3.3 Analysis and Discussion

According to the experiment results (cf. section 3.2) it is possible to conclude that all kinds of users were able to answer the simplest question. This is mainly because the information is accessible in only one resource (i.e. one page) and therefore it is quite straightforward to find the answer. For instance, most users answered this question using the traditional Wikipedia search. However, in order to correctly answer the most complex questions, SPARQL proficiency revealed to be necessary. This is because only these users were able to formally query the repository in order to obtain the necessary resources. However, even these users had some difficulties in understanding the underlying ontology to properly formulate the SPARQL query. For answering complex questions, a non-proficient SPARQL user that answered the complex questions 2 and 3 used the Wikipedia Search to randomly access resources in order to find if it fulfills the questions' constraints. This type of search is time consuming and uncomfortable. Moreover, users that tried to use gFacet and RelFinder did not find any answer at all.

Further, the evaluation of the proposed tools based on the correctness of the given answers shows that complex questions were not successfully answered by all kinds of users. Accordingly, only SPARQL proficient users answered all the questions. Hence, even the simplest question was wrongly answered by 2 users that used the Wikipedia Search to access the resource page and find the wanted result. This highlights that human error is likely to happen and that machine-supported systems may produce better results.

Finally, the results show that users can only answer complex questions in reasonable time if using the SPARQL tool. However, not all users are comfortable with the SPARQL language. Furthermore, even if familiar with the SPARQL language, users must have to know the underlying ontology of the repository. However, this is not usually the case.

In that sense, we may conclude that the repositories' front-ends do not provide enough support for the question answering task. On the one hand, the more accessible tools for non-experienced users (e.g. Wikipedia Search) do not provide the necessary means to answer complex questions. On the other hand, the most specialized tools (e.g. Virtuoso SPARQL Query Editor) are not accessible for all types of users. Hence, it is obvious that questions like the ones used in the experiment are not answerable through "search, browsing & navigation" operations without further data processing. Accordingly, it is possible to conclude that:

- Non-proficient users are not capable of responding such kind of questions. For instance, these users do not know how to formulate SPARQL queries;
- Proficient users eventually may respond such kind of questions but they have to use more than one tool to achieve the desired results. Moreover, they take some time and a lot of effort to understand the repository's underlying schema/ontology.

Based on these findings, it is possible to conclude that current search tools revealed to be insufficient to answer such questions because none of them contains all the necessary "basic" functionalities. Basic functionalities are the ones that are considered needed to help (non-proficient and proficient) users to answer complex questions.

Additionally, the answers given in the questionnaire reinforce the idea that state-of-the-art tools do not provide the best support while answering complex questions. Accordingly, users stated that only the Virtuoso SPARQL Query Editor provide the means to answer complex questions while all the other tools proved to be inadequate.

Moreover, the surveyors asked the users to express the limitations encountered with the used tools and how they would like to be assisted in their task. Accordingly, users stated that the user should have a more important role in the search task. Besides, the tools user interface should be enriched (e.g. adding semantic information) and its performance should be enhanced. Hence, the users' feedback resulted in the following which-list of features:

- Help find the attributes names whose values include a specific value;
- Provide abstractions for class expressions (e.g. $+10MPopulatedCity = City \cap \exists.populationtotal(> 10M)$);

- Abstract the subsumption relation (e.g. the $City \rightarrow PopulatedPlace \rightarrow Place$), such that it is possible to filter the individual of the super class to those of the subclass;
- Reduce the size of the proposed ontology entities, based on the current query context (defined by the ontology entities already selected);
- Suggest/Identify ontological objects based on a text search;
- Find all the/a set of relationships between two given ontological objects;
- Sort the retrieved list of relationships according to the number of occurrences;
- Retrieve the most common values of a certain objects' property;
- Translate the users' built question to a formal query supported by the repository;
- Query the repository with the built question and see the results;
- Combine all these requirements into a single tool.

3.4 Concluding Remarks

Preliminary experiments on state-of-the-art applications revealed that it is desirable to develop new applications that properly support the users in the information navigation and search tasks thereby improving their complex querying task. According to the experiment results and observations, it is envisaged a set of functionalities that should be included in a complex-query supporting front-end. Consequently, an application that provides such functionalities must be developed.

4 The Proposal

This chapter describes the thesis proposal and is organized as follows:

- First, a solution to solve the problems identified in the preliminary experiments is briefly described;
- Second, the encountered solution is deepened and formally described.

4.1 Informal Overview

Based on the preliminary experiments described in Chapter 3, different components were defined to answer the users' most basic needs in information navigation and search tasks that ultimately leads to answering complex questions. Figure 13 depicts the proposed solution.

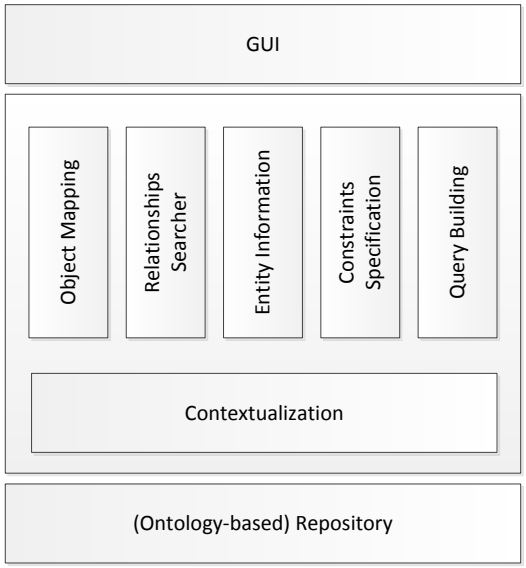

Figure 13 – System architecture

The proposal relies on a set of eight high-level components that are suitably combined into a user application. These components are summarized as follows.

- GUI is responsible for the interaction with the user: (i) it enables the user to formulate queries, and (ii) presents the respective responses (i.e. the retrieved resources).
- The Object Mapping serves to make the mappings between the (free) text inserted by the user and the corresponding ontology entities in the repository;
- Relationship Search is responsible for finding relationships between elements in the repository;
- Entity Information is the module responsible for retrieving some information related to an element in the repository;
- Constraints Specification is used to create constraints and, therefore, filter the query results;
- Query Building is responsible for automatically creating an appropriate formal query supported by the repository;
- Contextualization is the module responsible for narrowing the search area to the user current question' context;
- (Ontology-based) Repository is where the data/information is maintained. Notice that these repositories should be explicitly or implicitly described by ontologies.

Through this application the user interacts and exploits the functionalities of such components iteratively and arbitrarily in order to obtain a set of results to the complex question s/he is trying to answer (i.e. to build the query). Accordingly, the user should guide the search in several interactions with the system based on her/his necessities. Therefore, it is implied that the proposal consists in an assisting tool for iteratively, incrementally, and interactively navigate and retrieve information from repositories described by ontologies. It is iterative because the query building process phases are repeated several times (iterations). Further, it is incremental because the query is being progressively built and refined along the iterations. Finally, it is interactive because the user is requested to participate in the process of building/refining the query.

Although it is seen as a critical component in the system, this work does not focus on matters related with the GUI component. Moreover, it was assumed that the ontology-based repository already exists and it is available for the intended purposes. Therefore, the next section only specifies the remaining components.

4.2 Formal Specification

This section formally describes the middle ware components of the envisaged solution (i.e. Object Mapping, Relationship Search, Entity Information, Constraints Specification, Query Building, and Contextualization) in the form of an Application Programming Interface (API) thus being easily integrated in any system. In that sense, the envisaged API is deeply explained and formally defined.

Following sub-sections formally describe each one of these five API components.

4.2.1 Object Mapping

The Object Mapping component is responsible for mapping a natural language text introduced by the user to ontological entities. This is the most primary component as it is the only one that allows the user to inquiry the system in natural language for finding. Accordingly, this is where the user usually starts her/his quest for information.

For that, it is envisaged a three steps approach as follows.

The first step is completely automatic. It consists on applying a set of NLP techniques (e.g. tokenization, stops words, lemmatization, stemming) to identify the relevant terms in a natural language text thus improving the terms automatic detection (cf. **Example 1**). Moreover, the usage of a thesaurus might also improve this task by retrieving the inserted text related vocabulary (e.g. synonyms). This step is formalized as follows.

Definition 8 (Terms Identification) – The terms identification task is seen as a function $nlp\colon text \rightarrow \mathcal{T}$ such that $text$ refers to a set of words/phrases according to a natural language and $\mathcal{T}$ is the set of relevant terms resulting from a linguistic interpretation of $text$.

Example 1 – Imagine the situation where the user is asked to answer the **Sample Question 1** (i.e. *All soccer players, who played as goalkeeper for a club that has a stadium with more than 40.000 seats*). As the user does not know the repository schema, s/he might want to start by asking the system which entities are related to the text "*soccer player*" as it is mentioned in the question. Thus, the Object Mapping component is required to map such text to some ontology entities such that $nlp("soccer\ player") \rightarrow \mathcal{T}_1\colon \mathcal{T}_1 = \{soccer, player\}$.

The second step is also completely automatic. It aims to identify for each term the set of plausible ontology entities the user is interested in (cf. **Example 2**). In order to constrain the mappings to the relevant ontological entities the search context should be provided. This context is obtained through the application's state that is based on the users' interactions with the system. As consequence of the first step, the list of relevant terms is expanded thus providing a more complete vocabulary to map. This step is formalized as follows.

Definition 9 (Mapping Terms) – The task of mapping a set of terms ($\mathcal{T}$) to the corresponding ontological objects is a function $map(\mathcal{T}, \mathcal{O}, \mathcal{C}) \rightarrow Map$ such that:

- $\mathcal{T}$ is a set of terminological terms;
- $\mathcal{O}$ is the ontology describing the repository;
- $\mathcal{C} \subseteq \mathcal{O}$ is a sub-set of the ontology describing the repository. It define the ontological context in which the terms introduced by the user must be first considered;
- Map is the set of mappings found. Each element $m \in Map$ is a triple $m = (t, e, v)$ expressing that the term $t \in \mathcal{T}$ is mapped to the ontological entity $e \in \mathcal{O}$ (or $e \in \mathcal{C}$ if $\mathcal{C}$ is provided) with a confidence value $v \in]0, \infty[$. The ontological entity might be a class, a property, an individual or an axiom (e.g. a constraint).

Example 2 – In the second step, the two identified terms (*soccer* and *player*) must be mapped to ontological entities. Since there is no previous context for these terms, it is assumed that the context is all the ontology ($\mathcal{C}_1 = \mathcal{O}_{DBPedia}$). Accordingly, the system executes the $map(\mathcal{T}_1, \mathcal{O}_{DBPedia}, \mathcal{O}_{DBPedia}) \rightarrow Map_1$ function. Table 7 depicts the results of the Map_1 results where the confidence value of each pair term-entity is omitted by clarity reasons.

Table 7 – The ontological entities for the "*soccer*" and "*player*" terms

$\mathcal{T}$	**List of Entities**
soccer	{SoccerManager, SoccerClub, SoccerPlayer, SoccerLeague}
player	{GolfPlayer, SoccerPlayer, playerInTeam, TennisPlayer, BasketballPlayer}

The last step is the disambiguation. This consists of mapping each term $t \in \mathcal{T}$ to the unique ontological entity the user is interested in based on the outcome of the previous step (Map_1) (cf. **Example 3**). Despite the provided contextualization support, for each term $t \in \mathcal{T}$ might exist in Map more than one possible ontological entity to map. In such cases, and in order to reduce the user effort in selecting the ontological entities, a (semi-) automatic approach must be adopted.

Example 3 – In the disambiguation step, the system may strongly suggest the concept "*SoccerPlayer*" because it is the one common to both terms. However, the user may be able to select one of the other possible entities (Map_1 entities). Regarding the question in hands, consider that the user confirms the system suggestion' "*SoccerPlayer*". Accordingly, this entity is now the focus of the search.

Figure 14 depicts a mockup of a possible GUI for the Object Mapping component. In (1) is illustrated the text to search. Moreover, the results of the map function are shown in (2). Finally, the "*SoccerPlayer*" entity is selected thus demonstrating the (manual) disambiguation step.

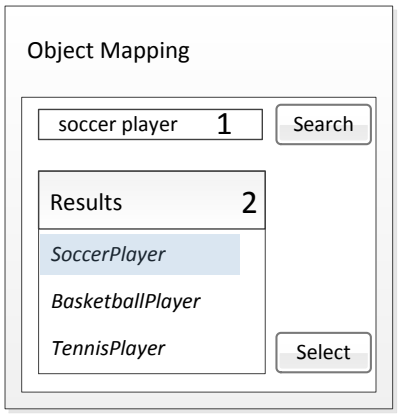

Figure 14 – Object Mapping GUI mockup

4.2.2 Relationship Searcher

The Relationship Searcher component is responsible for finding the ontological relations existing between two ontological entities in a given context. Once again, the context is automatically generated based on the application status. In this way, one can reduce the number of relationships retrieved to those that are relevant to the task at hands (cf. **Example 4**).

The process of finding relationships is defined as follows.

Definition 10 (Finding Relationships) – The task of finding relationships between two ontological entities is a function $relSearch(e, e', \mathcal{O}, \mathcal{C}, length, top, dir) \rightarrow Rels$ such that:

- $e \in \mathcal{O}$ and $e' \in \mathcal{O}$ are the ontological entities among which is necessary to find out the existing relationships;
- $\mathcal{O}$ and $\mathcal{C} \subseteq \mathcal{O}$ are the ontology describing the repository and the ontological context defining the searching space respectively;
- $length \geq 1$ defines the maximum admissible number of entities between e and e'. If $length = 1$, it means that only direct relationships between e and e' must be considered;
- $dir \in DIR$ specifies directionality constraints to the relationships. As an example, considering $DIR = \{forward, backward\}$, one might constraint relationships to those that goes from e to e' only ($forward$) or vice-versa ($backward$);
- top determines the amount of most common relationships to retrieve;
- $Rels$ is the set of relationships existing between e and e'. Each element $rel \in Rels$ is a tuple $rel = (r, v)$ such that r is a relationship path either in the form of $\{e, e_1, e_2, \dots, e_n, e'\}$ or $\{e', e_1, e_2, \dots, e_n, e\}$ where $n \leq length$ and $v \in]0, \infty[$ is a value expressing the relevance of the relationships path.

Example 4 – After finding the "*SoccerPlayer"* and "*SoccerClub"* entities through the Object Mapping component (cf. section 4.2.1), the question in hands (i.e. **Sample Question 1**) suggests that there is a relationship between these entities. However, the user typically does not know such relationship. Therefore, users might benefit from mechanisms that automatically find relationships in the repository schema based on two given entities that s/he knows. Notice that the context $\mathcal{C}_1$ is obtained from the current application status that is obtained according to the user' selected entities ("*SoccerPlayer*" and "*SoccerClub*"). Accordingly, the user could request the system to find the top five direct and straightforward relationships between "*SoccerPlayer"* and "*SoccerClub"* using the $relSearch(\mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: SoccerClub, \mathcal{C}_1, 1{,}5, forward) \rightarrow Rels_1$ function. The result of the function $Rels_1$ is depicted in Table 8.

Notice that these relationships are usually described by ontological properties and therefore the user is usually interested in the properties that relate the entities. Accordingly, the system should be able to retrieve the $Rels_1$ (2) that relate *"SoccerPlayer*" and "*SoccerClub*" (1) as depicted in Figure 15.

Table 8 – The five most relevant relationships between *"SoccerPlayer"* and "*SoccerClub*"

r	v
{SoccerPlayer, clubs, SoccerClub}	0.95
{SoccerPlayer, club, SoccerClub}	0.90
{SoccerPlayer, team, SoccerClub}	0.85
{SoccerPlayer, currentclub, SoccerClub}	0.60
{SoccerPlayer, youthclubs, SoccerClub}	0.50

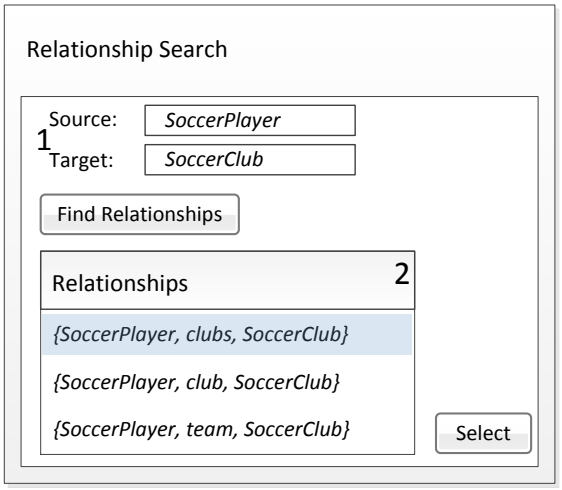

Figure 15 – Relationship Search GUI mockup

In order to continue the walkthrough example, let us admit that the user selects the property "clubs" as suggested by the system.

4.2.3 Entity Information

The Entity Information component is responsible of showing some information related with a specific entity to the user. Accordingly, the user is able of finding the properties and its values of an ontological entity s/he is interested in. For that, two processes were identified.

The first process consists on finding the most common properties of a given ontological entity in a given context thus helping the user understanding which are the attributes of a given entity (cf. **Example 5**).

This process is formalized as follows.

Definition 11 (Entity Common Properties) – The task of determining the most common properties of an entity is a function $properties(e, \mathcal{O}, \mathcal{C}, top) \rightarrow P$ such that:

- $e \in \mathcal{O}$ is the ontological entity to retrieve the most common properties;
- $\mathcal{O}$ and $\mathcal{C} \subseteq \mathcal{O}$ are respectively the ontology describing the repository and the ontological context for determining the common values of p;
- top determines the amount of most common properties to retrieve;

- P is the set with the top most common properties of e. Each element $p \in P$ is a tuple $p = (p', v)$ such that p' is an ontological property and $v \in]0, \infty[$ is a value expressing how common p' is regarding to e.

Example 5 – For answering **Sample Question 1**, the user now needs to find the "*SoccerPlayer"* property that allows restricting to only those that plays as goalkeepers. Accordingly, as the user does not know the DBPedia ontology, one way to do this is by asking the system which are the most common properties of the "*SoccerPlayer"* entity. According to the application status, the user is now focused on the *"SoccerPlayer*", "*SoccerClub*" and "*clubs*" entities. Therefore, the user may request the system to retrieve the "*SoccerPlayer*"' top five common properties through the $properties(\mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}, \mathcal{C}_2, 5) \rightarrow P_1$ function. The obtained results are depicted in Table 9:

Table 9 – The five most relevant attributes for *"SoccerPlayer*"

p	v
clubs	0.90
teams	0.85
position	0.80
club	0.60
currentclub	0.50

These results may be represented as depicted in Figure 16. Moreover, the user may now select the $position$ property which is probably the one that s/he is interested in.

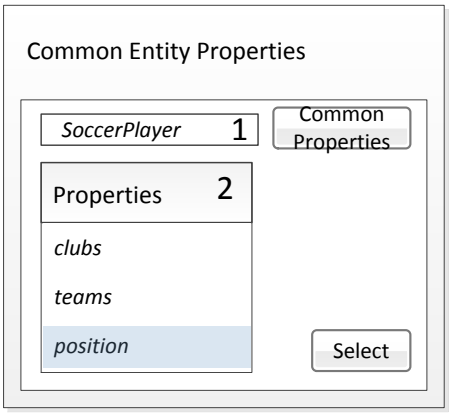

Figure 16 – Common Properties GUI mockup. (1) is the entity to retrieve the related properties (2)

The second process consists on retrieving the values of the properties (i.e. attributes) of an ontology entity (cf. **Example 6**). This process allows the user to effectively access the entity attribute values thus helping the user answering some (simple) questions related with a given entity.

Definition 12 (Entity Property Values) – The task of retrieving the value(s) of a property in the perspective of an entity is a function $values(e, p, \mathcal{O}, \mathcal{C}, max) \rightarrow PV$ such that:

- $e \in \mathcal{O}$ is the ontological entity to retrieve the property values;
- $p \in \mathcal{O}$ is the ontological property to retrieve the values;
- $\mathcal{O}$ and $\mathcal{C} \subseteq \mathcal{O}$ are respectively the ontology describing the repository and the ontological context for determining the common values of p;
- max determines the maximum amount of property values to retrieve;
- PV is the set with the max values of property p related to e. Each element $pv \in PV$ is a property value (either an ontological entity or a literal).

Notice that it is possible to retrieve all the attributes of a given entity by the combination of the two described processes. Accordingly, the product $P \times PV$ returns all the property values of an ontological entity.

Example 6 – Now, if the question asked the user to retrieve the position of the Cristiano Ronaldo player (e.g. *In which position does Cristiano Ronaldo plays?*) the user could ask the system to retrieve the value of the "*position*" property (2) for the "*Cristiano Ronaldo*" entity (1). Accordingly, the user should first search for the "*Cristiano Ronaldo*" entity using the Object Mapping functionality (cf. section 4.2.1). After finding the *"Cristiano Ronaldo*" entity, the user is able to question the system. For that, the user could use the $values(\mathcal{O}_{DBPedia}: Cristiano\ Ronaldo, \mathcal{O}_{DBPedia}: position, \mathcal{O}_{DBPedia}, \mathcal{C}_2, 1)$ function such that the result should be the entity "*Forward*" as depicted in Figure 17.

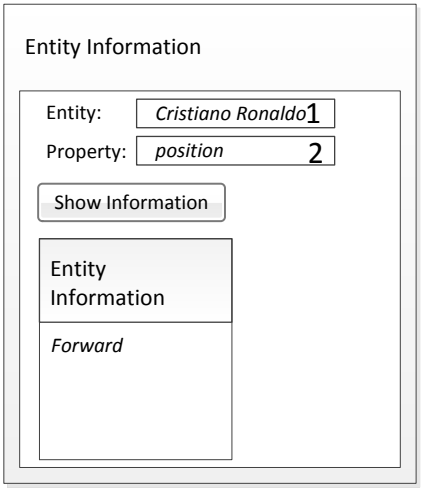

Figure 17 – Cristiano Ronaldo position example

4.2.4 Constraints Specification

The Constraints Specification component is responsible for supporting the user to specify constraints over the ontological entities (properties) s/he is interested in. By this means, the user is able to properly filter the retrieved results such that they meet the user's needs and abstract (identify) such class of individuals.

For that, three processes were identified.

The first process consists on determining which logical operators (e.g. equal, greater than, contains) are applicable to a given ontological property considering both the existing ontological definitions (e.g. the range) and the context on which that property is being used (cf. **Example 7**). The context, which is automatically generated according to the status of the application, allows restricting the values that the property might take to only those in the current context. Consequently, it does not return operators that are not applicable to both the property and the context at hand. This process is formalized as follows.

Definition 13 (Applicable Operators) – The task of determining the applicable logical operators on a property is a function $opers(p, \mathcal{O}, \mathcal{C}, \Upsilon) \rightarrow \Upsilon'$ such that:

- $p \in \mathcal{O}$ is the ontological property to determine the applicable operators;
- $\mathcal{O}$ and $\mathcal{C} \subseteq \mathcal{O}$ are the ontology describing the repository and the ontological context on which p is used respectively;
- Υ is the set of available logical operators in the system;
- $\Upsilon' \subseteq \Upsilon$ is the sub-set of available logical operators that are applicable to p in the context $\mathcal{C}$.

Example 7 – At this point, the user identified the "*SoccerPlayer*" and "*SoccerClub*" entities and a relationship between them (i.e. the property "*clubs*"). Moreover, the user also noticed that the "*position*" property may relate the player with his field position. This specification is depicted in Figure 18.

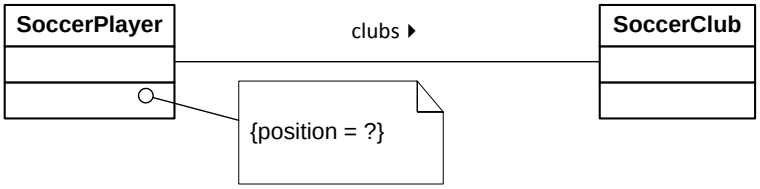

Figure 18 – Conceptual specification made by the user

Now, in order to answer **Sample Question 1**, the user needs to find all the "*SoccerPlayer*" that plays as goalkeeper. However, the user does not know how to express such relationship. In that sense, the user might ask the system which are the applicable operators for the "*position*" property. Moreover, as the context did not changed since the previous example (the application status is the same) it is assumed that $\mathcal{C}_3 = \mathcal{C}_2$. Therefore, the user might inquire the system by the $opers(\mathcal{O}_{DBPedia}{:}\, position, \mathcal{O}_{DBPedia}, \mathcal{C}_3, \Upsilon) \rightarrow \Upsilon_1$ function such that Υ_1 represents the applicable operators to the "*position"* property which are depicted in Table 10.

Table 10 – The applicable operators to the property "*position*"

p	Range	Υ_1
position	string	{=, ≠, contains, starts, ends}

The second process consists on retrieving the most common values taken for a property in a given context. This process aims to help the user on:

- Perceiving which kind of values are admissible (e.g. numerical, string, date, ontological resource);
- Typing the admissible values. This is especially helpful for (object) properties whose admissible values are ontological entities (resources) only. In this case, it eases the required object mapping task since the most common values are themselves ontological entities.

This process is formalized as follows.

Definition 14 (Property Common Values) – The task of determining the most common values of a property is a function $values(p, \mathcal{O}, \mathcal{C}, top) \rightarrow PV$ such that:

- $p \in \mathcal{O}$ is the ontological property to retrieve the most common values;
- $\mathcal{O}$ and $\mathcal{C} \subseteq \mathcal{O}$ are respectively the ontology describing the repository and the ontological context for determining the common values of p;
- top determines the amount of most common values to retrieve;
- PV is the set with the top most common values of property p. Each element $pv \in PV$ is a tuple $pv = (pv', v)$ such that pv' is a property value (either an ontological entity or a literal) and $v \in]0, \infty[$ is a value expressing how common pv' is regarding to p.

Example 8 – However, the returned values in **Example 7** do not provide any hint on the existing property values. In this sense, the user may not envision that exists a "Goalkeeper" string representation to denote the goalkeeper field position. Therefore, one way to overcome this problem is to ask the system which are the most common values for the "*position*" property. Once again, the application status did not changed and therefore $\mathcal{C}_4 = \mathcal{C}_3$. In this sense, the user calls the $values(\mathcal{O}_{DBPedia}: position, \mathcal{O}_{DBPedia}, \mathcal{C}_4, 5) \rightarrow PV_1$ function. Table 11 depicts the PV_1 results.

Table 11 – The five most common values of the property "*position*"

p	pv'	v
position	Midfielder	0.85
	Defender	0.80
	Striker	0.75
	Goalkeeper	0.70
	Forward	0.60

Further, for simplifying purposes it is assumed that the user has found the property "*capacity*" that allows to restrict the clubs to those that have a stadium with more than 40 0000 seats.

At this point, the user has refined his specification as depicted in Figure 19. According to this specification, the user is now able to properly construct the query to answer the question in **Sample Question 1**.

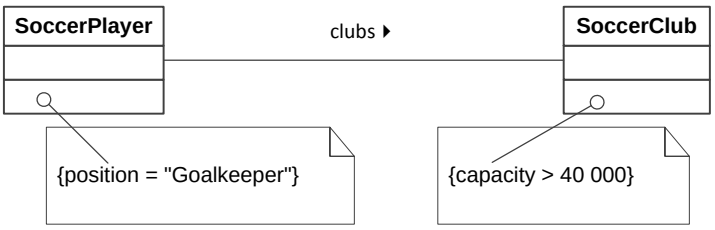

Figure 19 – The final specification made by the user

The third process concerns the ability to construct the appropriate constraint given (i) an class or an individual (ii) a property, (iii) an operator ($op \in OP$) and (iv) a property value (pv'). Accordingly, this is the process that actually allows the user to construct the constraint after having found all the necessary parameters. This process is formalized as follows.

Definition 15 (Constraint) – The task of constructing a constraint on a property is a function $constraint(p, v, pv') \rightarrow \lambda$ such that:

- $e \in \mathcal{O}$ is the ontological class or individual to apply the constraint;
- $p \in \mathcal{O}$ is the ontological property on which the constraint applies on;
- $v \in \Upsilon$ is the logical operator applied in the constraint;
- pv' is a property value;
- λ is the resulting constraint on p such that v is true for pv'.

Example 9 – Finally, the user has all the information needed to create the desired constraints. In this sense, the user may request the system to create a constraint such that $constraint(\mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: position, =, "Goalkeeper")$ as depicted in Figure 20.

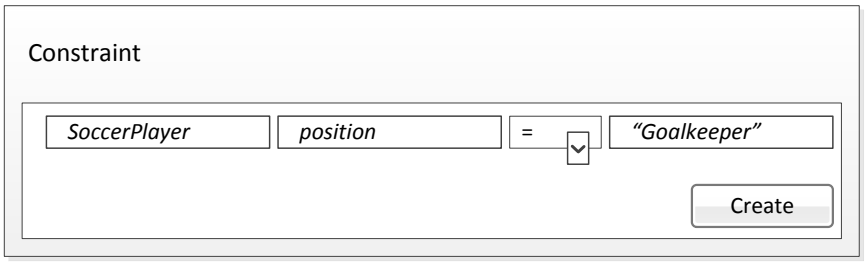

Figure 20 – Constraint GUI mockup

Furthermore, the user may create a constraint to filter the clubs by their stadium capacity such that $constraint(\mathcal{O}_{DBPedia}: SoccerClub, \mathcal{O}_{DBPedia}: capacity, >, 40\ 000)$.

At this point, the user has created all the necessary constraints to answer the question.

4.2.5 Query Building

The Query component is responsible for dynamically generating the appropriate query ($\mathcal{Q}$) in a query language (QL) supported by the repository (e.g. SPARQL) in order to execute it against the repository and, therefore, to retrieve the results (i.e. the answer) for the complex question formulated by the user (cf. **Example 10**).

This process is formalized as follows.

Definition 16 (Query) – The task of constructing a query is a function $query_{QL}(\mathcal{G}, \Lambda, \mathcal{Z}) \rightarrow \mathcal{Q}_{QL}$ such that:

- $\mathcal{G}$ is a connected graph composed by the ontology entities selected by the user;
- Λ is the set of constraints specified by the user;
- $\mathcal{Z}$ is a set of ordering clauses. Each element $z \in \mathcal{Z}$ is a tuple $z = (e, asc)$ such that $e \in \mathcal{G}$ is an ontological entity and $asc \in \{true, false\}$ stating an ascending ($true$) or an descending ($false$) order;
- $\mathcal{Q}_{QL}$ is the resulting query formulated in the query language QL and is ready to be executed against the repository.

Example 10 – The final step in the query formulation process – the process that allows to query a system in order to obtain answers – is the query construction. Accordingly, in order to effectively answer **Sample Question 1** the system should allow the user to automatically build the formal query based on the previously identified requirements (depicted in Table 12)

Table 12 – The question' requirements (i.e. the selected entities and constraints)

Selected Entities	{SoccerPlayer, SoccerClub, clubs, position}
Constraints	{<SoccerPlayer,clubs,SoccerClub>, <SoccerPlayer, position,=,"Goalkeeper">, <SoccerClub,capacity,>,40 000>}

Accordingly, the user asks the system to create a query based on the question requirements without any ordering clauses using the $query_{SPARQL}(selected, constraints, \emptyset) \rightarrow \mathcal{Q}_{SPARQL_1}$ function such that:

- $selected = \{\mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: SoccerClub, \mathcal{O}_{DBPedia}: clubs, \mathcal{O}_{DBPedia}: position\}$;
- $constraints = \{\langle \mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: clubs, \mathcal{O}_{DBPedia}: SoccerClub\rangle, \langle \mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: position = "Goalkeeper"\rangle, \langle \mathcal{O}_{DBPedia}: SoccerClub, \mathcal{O}_{DBPedia}: capacity > 40\ 000\rangle\}$.

Figure 21 depicts such scenario. In order to build the formal query (3), the system must take into account the users' selected entities (1) and constraints (2) and with these parameters it should be able to construct the formal query in order to query the repository and therefore show the results.

Figure 21 – Query Building exemplification

4.2.6 Contextualization

The Contextualization component is responsible for providing one or more relevant fragments of the ontology describing the repository according to a desired level of abstraction. Usually, this component is used by other components in the API in order to (i) restrict the search space thus decreasing the results to retrieve, and to (ii) suggest related elements in the repository that might be of interest. In this way, one might decrease the information overload helping the user to process the information and provide some contextualized information in order to focus on the desired context of the search.

By definition, ontology modularization (cf. section 2.2) is the ontology engineering process that provides such services. Accordingly, the contextualization component makes use of ontology modularization techniques (described in section 2.2) to achieve its main goal (i.e. reduce the size and/or complexity of the knowledge schema).

In order to properly capture the (search) context, it is necessary to take into account the users' interaction with the system. Accordingly, the context needs to be extracted according to the application status. This status is used to keep tracking of the ontology entities that the user is interested in (i.e. the focus of the search). One way of capturing the application status is through the inputs to the query function $query_{QL}(\mathcal{G}, \Lambda, \mathcal{Z})$ since it represents the users' focus. Accordingly, $\mathcal{G}$ contains the users' selected entities and Λ the set of constraints applied to those selected entities. Then, using some modularization techniques the system is able to properly capture the search context. For instance, using the module extraction function (cf.

Definition 5), the system can extract a module that represents the search context. Moreover, using the ontology summarization function (cf. **Definition 7**), the system may retrieve an even smaller module (i.e. context) with some related elements to add to its search. An exemplifying scenario is depicted in **Example 11**.

Example 11 – Continuing **Example 10**, imagine that the user was not fully satisfied with the final results. Therefore, the user should continue her/his search using the system functionalities. However, the system could help the user in this task by suggesting some entities of the ontology that were related to the users' search. Accordingly, the system could extract a module $\mathcal{M}$ from $\sigma: (\mathcal{O}, \mathcal{S})$, being $\mathcal{S} = \{\mathcal{O}_{DBPedia}: SoccerPlayer, \mathcal{O}_{DBPedia}: SoccerClub, \mathcal{O}_{DBPedia}: clubs, \mathcal{O}_{DBPedia}: position\}$ (i.e. the application status). However, as the module $\mathcal{M}$ is very large to present to the user (e.g. contains more elements than a pre-defined number), the system automatically extracts a description $\mathcal{D}$ of $\mathcal{M}$ using the ontology summarization algorithm $\varphi: \mathcal{O} \rightarrow \mathcal{D}$ such that $\mathcal{D} = \{\mathcal{O}_{DBPedia}: Athlete, \mathcal{O}_{DBPedia}: youthclubs, \mathcal{O}_{DBPedia}: currentclub, \mathcal{O}_{DBPedia}: captain, \mathcal{O}_{DBPedia}: coach\}$. In this sense, a smaller representation of the context was extracted and can now be showed to the user as depicted in Figure 22.

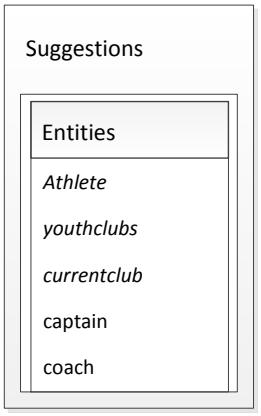

Figure 22 – Contextualization example

5 Development

This chapter describes the implementation efforts of the proposed ideas in the context of the World Search project. Accordingly, the implementation details of each component are succinctly described and a prototype is presented.

The components of our proposal integrate into the World Search architecture as illustrated in Figure 23. The proposal components have been implemented in a prototype denominated Complex Query Building (CQB). It is worth mentioning that the components integrated into the Ontological Services module are all stateless. Therefore, it is responsibility of the Semantic Query module to provide an application status. In this sense, the CQB prototype is a stateless API that provides a set of functionalities for information retrieval and query building over an ontology-based repository.

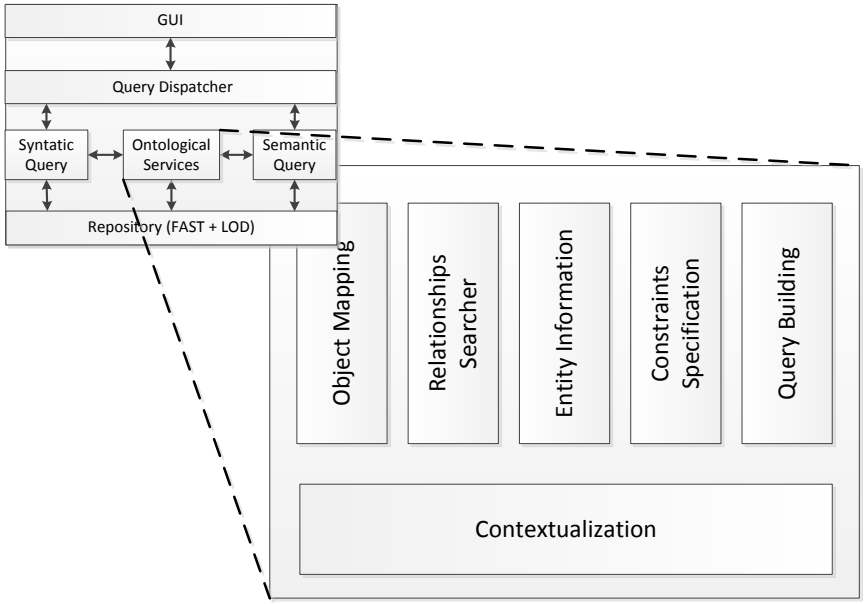

Figure 23 – The proposed components integrated in the World Search architecture

The CQB prototype was developed as a standalone Java application where each one of the identified functionalities is available.

Next, each one of the developed components is described.

5.1 Object Mapping

As in traditional search engines, object mapping makes use of indexing mechanisms to retrieve the entities/resources based on the inserted keywords. Accordingly, the suggestions provided by the Object Mapping component are retrieved by the Lucene [Lucene 2012] indexing mechanism. Lucene was adopted because it is free to use and also perform good results in comparison with FAST (that is a proprietary application). This mechanism indexes the content of a configurable set of selected properties determined by the entity type. Current configuration is represented in Table 13. Each selected property is associated to a weight-value, which is used for ranking purposes. Notice that, for testing purposes only small fragments of the DBPedia datasets were indexed.

Table 13 – Current indexed selected properties.

Entity Type	**Dimensions**			
	$\mathcal{T}$-box		**$\mathcal{A}$-box**	
	Property	**Weight**	**Property**	**Weight**
Class	rdf[1]:label	3	n/a	
	rdf:comment	1		
Object Property	rdf:label	3	-	-
	rdf:comment	1	-	-
Data Property	rdf:label	3	"value"	2
	rdf:comment	1		
Individuals	-	-	rdf:label	3

Moreover, the indexing mechanism makes use of a text analyzer which is responsible for a set of NLP tasks on the text content to be indexed. The same analyzer is also applied on the text introduced by the user in order to improve the suggestions made (i.e. the same analyzer is used on both tasks for producing similar results).

As described in **Example 1**, for answering the **Sample Question 1** the user would likely search for the text "soccer player" in order to find the corresponding entity in the repository. By doing this the user would likely find that the entity that s/he is interested in is *http://dbpedia.org/ontology/SoccerPlayer* as depicted in Figure 24.

[1] "rdf" stands for the RDF Schema vocabulary whose namespace is http://www.w3.org/1999/02/22-rdf-syntax-ns#

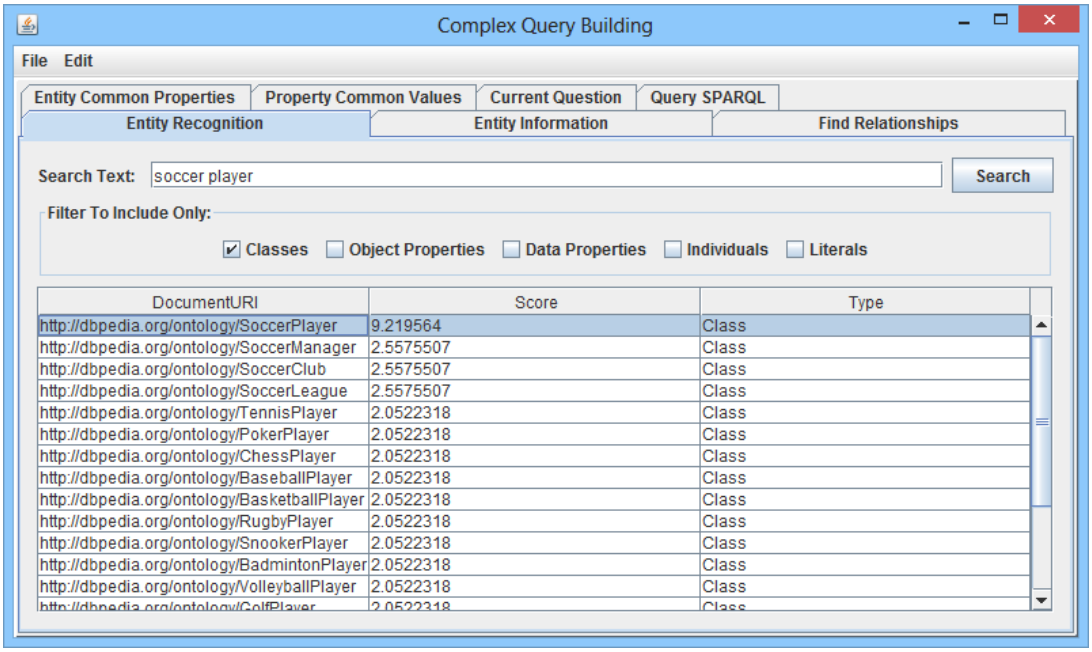

Figure 24 – "*SoccerPlayer"* entity recognition by text search input

Figure 24 depicts the user interface of the Object Mapping function in the CQB prototype. Notice that the Object Mapping GUI allows the user to specify/filter the type of entity s/he is interested in (e.g. Classes) thus discarding irrelevant information. In this case, the user only selected the classes of the ontology since "*Soccer Player*" it is likely being a concept. Moreover, the results table is divided in three columns: documentURI, Score, and Type. The documentURI column is where the URI of the result (e.g. resource, property, class) is shown. Moreover, the Score column provides a numeric value that shows the systems' confidence on how the results may match the inserted text (higher values reveal higher confidence). The type column shows the ontological entity' type (e.g. class, property, individual) thus allowing the user to perceive which kind of information s/he is dealing with. For instance, this might be helpful for selecting a resource when the resources have similar names.

5.2 Relationship Searcher

The Relationship Searcher component implementation relies on a set of pre-defined SPARQL queries to query the repository. The query to use is chosen according to the type of source and target entities. Accordingly, each pair of entity type requires a specific query. Therefore, as there are three different possible queries for each entity type (i.e. class, property or individual) the overall number of pre-defined queries is nine. However, the implemented algorithm only deals with direct relationships (i.e. one hop from the source entity to the target entity) and therefore only eight different queries were required. Moreover, the value expressing the relevance of each relationship path found is given by counting the number of times that relationship is instantiated on the context of the two input ontology entities. Further, the set of relationships found is pruned based on the input context.

According to **Example 4**, the user might need to know the 20 most significant relationships between the "*SoccerPlayer*" and "*SoccerClub*" entities. In this case, as both entities are of type Class, the generated query is as the one depicted in Figure 25.

```
select distinct ?rel (count(?rel) as ?value)
where
{
?a a <http://dbpedia.org/ontology/SoccerPlayer> .
?b a <http://dbpedia.org/ontology/SoccerClub> .
?a ?rel ?b .
}
GROUP BY ?rel
ORDER BY DESC (?value)
LIMIT 20
```

Figure 25 – Relationship Search sample query

The result of such task using the CQB tool is depicted in Figure 26.

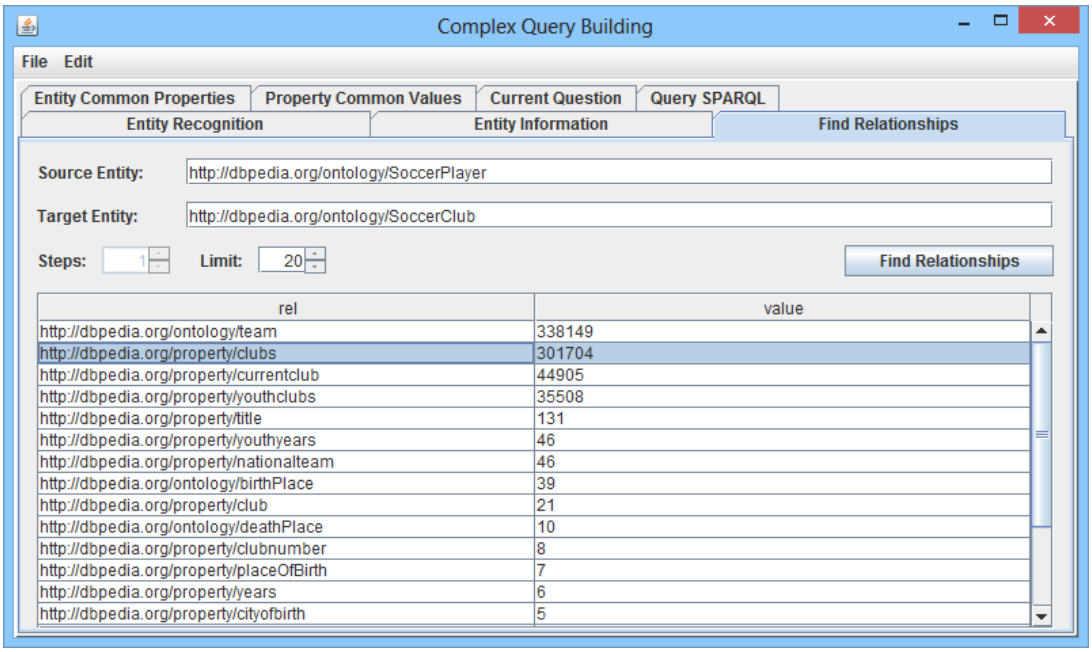

Figure 26 – Relationships between "*SoccerPlayer*" and "*SoccerClub*"

5.3 Entity Information

The Entity Information component relies on two pre-defined SPARQL queries against the repository. One query is for retrieving the most common properties of a given entity (cf. **Definition 11**) and the other query is for retrieve the property value(s) of an entity property (cf. **Definition 12**).

According to **Example 5**, when the user asks the system which are the properties related with the "*SoccerPlayer"* entity a query as the one depicted in Figure 27 is generated.

```
select distinct ?rel (count(?rel) as ?value)
where
{
?a a <http://dbpedia.org/ontology/SoccerPlayer> .
?a ?rel ?value .
}
GROUP BY ?rel
ORDER BY DESC (?value)
LIMIT 20
```

Figure 27 – Common Entity Properties sample query

The result of such task using the CQB tool is depicted in Figure 28.

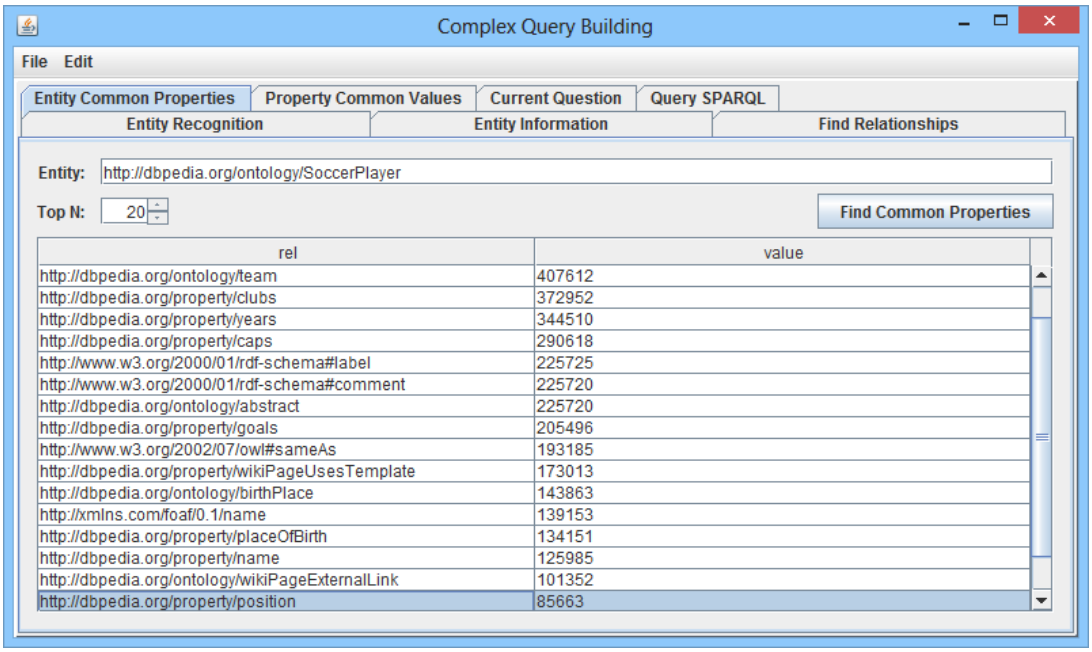

Figure 28 – Common "*SoccerPlayer*" properties using the CQB tool

Moreover, according to **Example 6** the user asks the system the specific attribute (i.e. property) value of a certain entity. In this case, the user asks the system which is the "*position"* property value of the "*Cristiano Ronaldo"* entity. In this sense, the system creates a query as depicted in Figure 29.

```
select *
where
{
<http://dbpedia.org/resource/Cristiano_Ronaldo> <http://dbpedia.org/property/position> ?value .
}
LIMIT 1
```

Figure 29 – Sample query for retrieving the Cristiano Ronaldo position

The results obtained using the CQB tool are depicted in Figure 30.

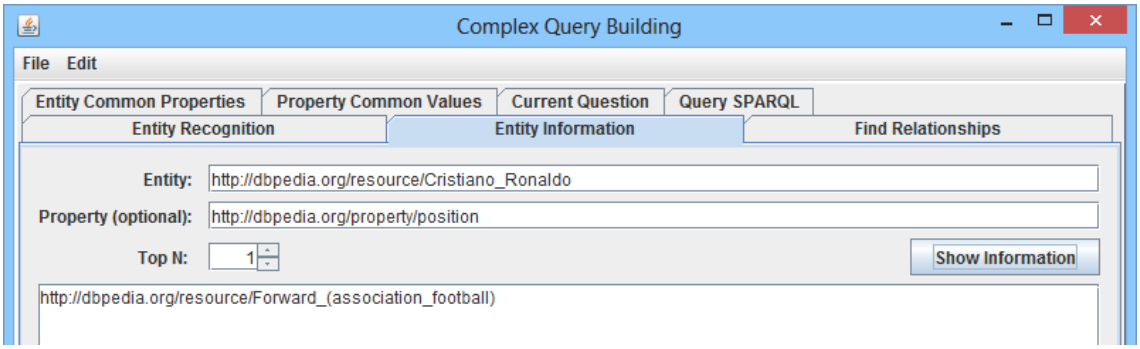

Figure 30 – "*position*" property value of the "*Cristiano Ronaldo*" entity

However, if the user wants to know all the attributes and the corresponding values of a certain entity, it should only provide the entity that s/he is interest in. This way, the system will perform the product $P \times PV$ that returns all the property values of an ontological entity thus creating a query as the one depicted in Figure 31.

```
select *
where
{
<http://dbpedia.org/resource/Cristiano_Ronaldo> ?property ?value .
}
LIMIT 100
```

Figure 31 – Sample query to retrieve all property values of a given entity

As depicted in Figure 32 the obtained results show 100 (because it is limited by the user) of the *Cristiano Ronaldo* properties and its values (e.g. his full name is Cristiano Ronaldo dos Santos Aveiro).

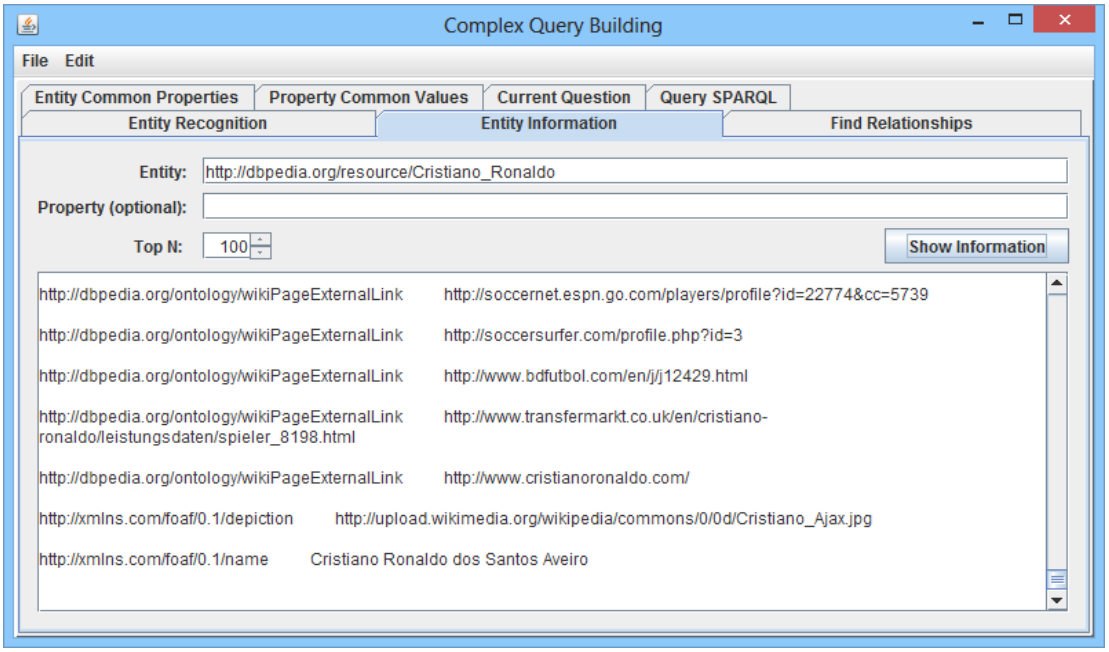

Figure 32 – "*Cristiano Ronaldo*" information

5.4 Constraints Specification

The Constraints Specification component makes use of an interpretation function mapping the range of a given property to a set of logical operators supported by the SPARQL endpoint of the repository. However, this functionality is not fully implemented yet as it only supports a set of pre-defined operators and only verifies if the user input is valid to the chosen operator.

The most common values of a given property are determined through a set of parameterized SPARQL queries against the repository. These queries might take into consideration the intended context on which the property is being used. Accordingly, two different pre-configured queries had been created. One does not take into account the context while the other does.

In the sequence of **Example 8**, using the CQB tool the user should be able to ask for the common values of the property "*position*" that is an attribute of the "*SoccerPlayer*" entity. Figure 33 shows the generated query.

```
select distinct ?value(count(?value) as ?rank)
where
{
?a <http://dbpedia.org/property/position> ?value .
?a a <http://dbpedia.org/ontology/SoccerPlayer> .
}
GROUP BY ?value
ORDER BY DESC (?rank)
LIMIT 20
```

Figure 33 – Sample context-aware query for retrieve common property values

Moreover, Figure 34 displays the obtained results using the CQB tool GUI.

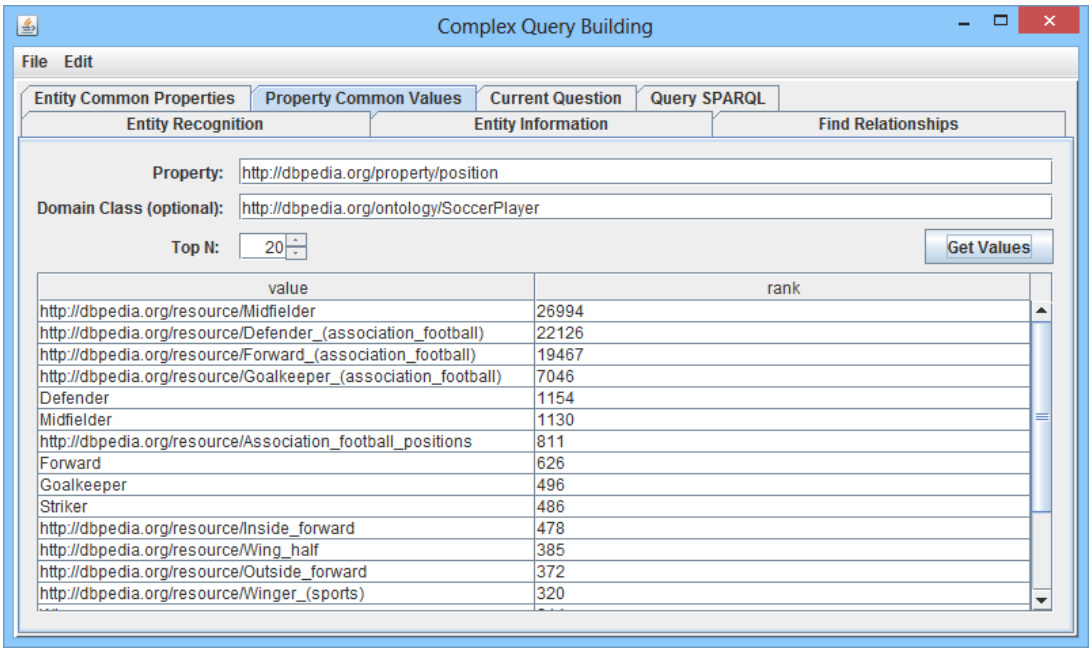

Figure 34 – "*Soccer Player*"'s "*position*" common property values

However, if no domain class was provided, the generated query (cf. Figure 35) and the results (cf. Figure 36) would be different.

```
select distinct ?value(count(?value) as ?rank)
where
{
?a <http://dbpedia.org/property/position> ?value .
}
GROUP BY ?value
ORDER BY DESC (?rank)
LIMIT 20
```

Figure 35 – Sample query for retrieve common property values

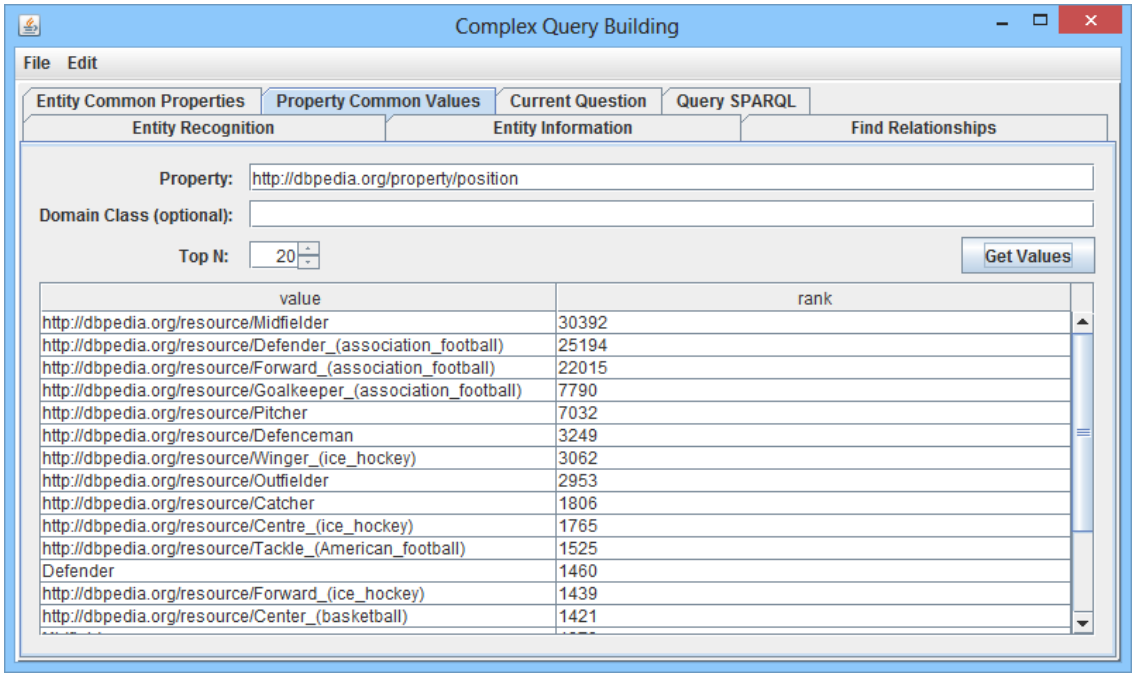

Figure 36 – "*position*" common property values

Thus, this is a scenario where the context exploitation clearly benefits/improves the quality of the output. It also reveals that the property "*position*" is used in more than one context/domain classes.

Finally, the user may create the constraint as depicted in Figure 37 (cf. **Example 9**).

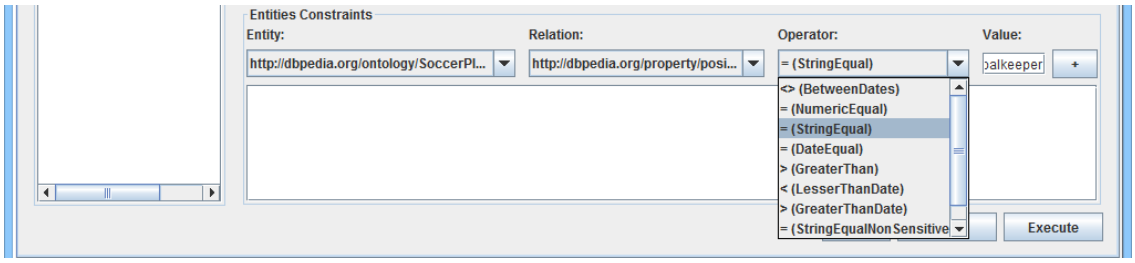

Figure 37 – Constraint creation example

5.5 Query Building

The Query Building component is implemented following a straightforward translation approach of a connected graph to a text representing a query in SPARQL.

Following the **Example 10**, the user is now able to properly formulate the desired question in order to query the repository and therefore properly answer **Sample Question 1**. Figure 38, shows the GUI panel that allows the creation of the questions using the CQB tool GUI.

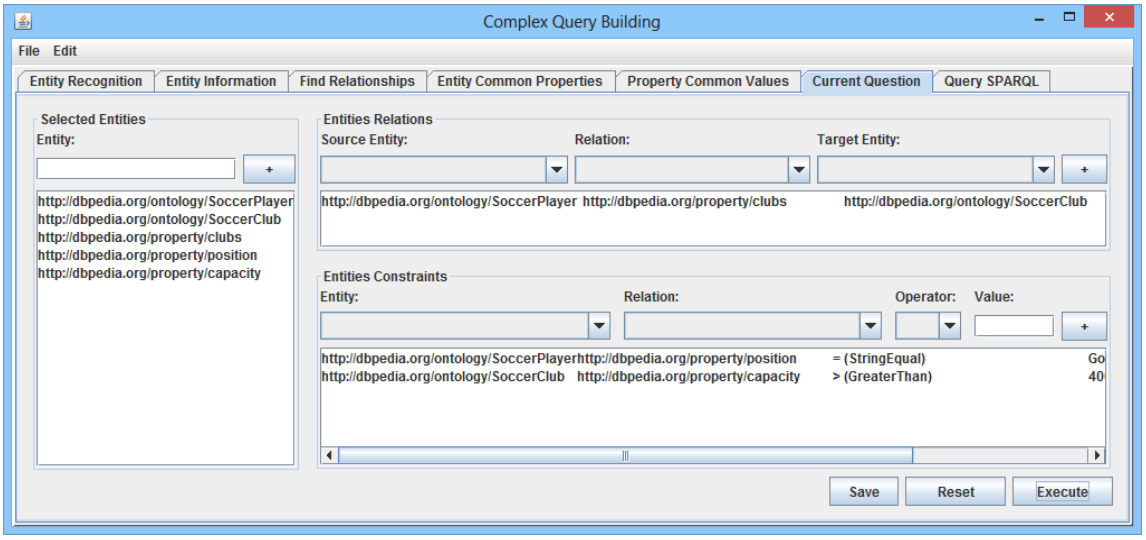

Figure 38 – Query Building Panel

The panel is divided in three main components. Selected Entities allows the user to insert new entities that s/he is interested in. Moreover, the constraints creation is divided in two components: (i) Entity Relations allow the restriction of the results based on relations between ontological entities (e.g. *SoccerPlayer:clubs→SoccerClub*) while (ii) Entity Constraints allows the creation of constraints based on the values of the (data) properties (e.g. *SoccerClub:capacity* > 40000).

When the user presses the "Execute" button, the system dynamically creates a SPARQL query that is used to query the DBPedia repository and shows the query results as depicted in Figure 39.

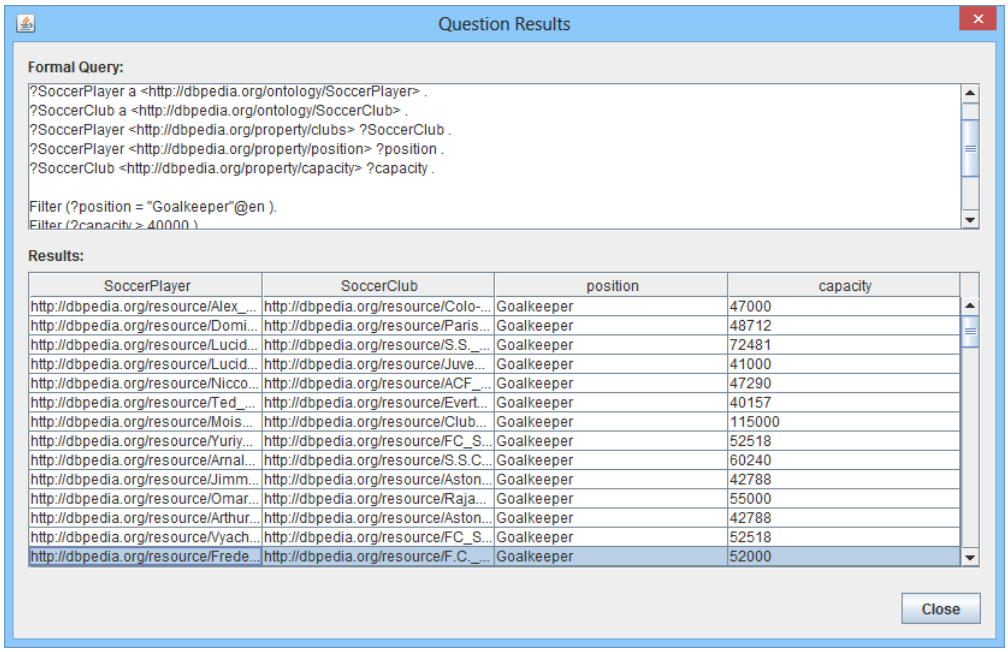

Figure 39 – Query results (i.e. the answer)

The full generated SPARQL query is shown in Figure 40.

```
Select *
{
?SoccerPlayer a <http://dbpedia.org/ontology/SoccerPlayer> .
?SoccerClub a <http://dbpedia.org/ontology/SoccerClub> .
?SoccerPlayer <http://dbpedia.org/property/clubs> ?SoccerClub .
?SoccerPlayer <http://dbpedia.org/property/position> ?position .
?SoccerClub <http://dbpedia.org/property/capacity> ?capacity .
FILTER (?position = "Goalkeeper"@en ) .
FILTER(?capacity > 40000) .
}
LIMIT 1000
```

Figure 40 – Final SPARQL query

5.6 Contextualization

In order to implement the contextualization component, the adoption of some modularization algorithms was envisaged. In this context, experiments with state-of-the-art modularization algorithms in [Brandão, Maio, and Silva 2012b] highlighted the need of some improvements to such algorithms. Accordingly, the contextualization component of the CQB prototype was deployed making use of a newly developed module extraction algorithm.

Algorithm 1 describes the developed module extraction algorithm, which for a given ontology $\mathcal{O}$ and a set of user's selected entities called seed signature $seed\mathcal{S}$, it extracts a contextualized module $\mathcal{M}$ such that $\mathcal{M} \subseteq \mathcal{O}$. For that, the input signature ($seed\mathcal{S}$) is first expanded through the Algorithm 2. The result is a new and expanded signature $\mathcal{S}$. Next, it is extracted from the ontology all the local axioms (Ax) related with the entities mentioned in the expanded signature. For that, it is used the syntactic local axiom extraction algorithm available in the OWL-API [OWL API 2009]. Moreover, for each ontological entity in $\mathcal{S}$ the algorithm extracts all the corresponding referencing axioms ($refAx$) using the *ReferencingAxioms* function from the OWL API and adds them to the final set of axioms Ax. Finally, the resulting module $\mathcal{M} = Ax$ is returned.

Algorithm 1. Module Extraction
Require: An ontology $\mathcal{O}$ and a signature $seed\mathcal{S}$
Ensure: A (contextualized) module $\mathcal{M}$ is provided.

1: $\mathcal{S} = SignatureExpansion(\mathcal{O}, seed\mathcal{S})$
2: $Ax = LocalAxioms(\mathcal{O}, \mathcal{S})$
3: **for all** $e \in \mathcal{S}$ **do**
4: $\quad refAx = ReferencingAxioms(e, \mathcal{O})$
5: $\quad Ax = Ax \cup refAx$
6: **end for**
7: $\mathcal{M} = Ax$
8: **return** $\mathcal{M}$

Algorithm 2 expands the input signature ($seed\mathcal{S}$) by adding other entities that are related to the entities in the input signature. The result is an expanded signature $\mathcal{S} \supseteq seed\mathcal{S}$. To determine the entities that are related with an entity of the input signature it is adopted four distinct behaviors which depend on the entity type. The isA function receives an entity as input and returns the type of that entity. The different behaviors of the algorithm depend on the entity type and are described as follows:

- if the entity (e) in the signature is a class, the algorithm starts by adding all it's the sub-classes ($getSubClasses$) to the expanded signature ($\mathcal{S}$) and then it adds all the direct (only one level above) super-classes ($getSuperClasses$);
- if the entity is an object property, all its domains ($getPropertyDomains$) and ranges ($getPropertyRanges$) are added to the signature $\mathcal{S}$;
- if the entity is a data property, the algorithm adds all its domains to $\mathcal{S}$;
- if the entity is an individual, the algorithm adds all its types ($getTypes$) to the signature $\mathcal{S}$ and then the types of the related individuals (e.g. individuals that are related by object property assertions) – by the $getRelatedInviduals$ – function are also added to $\mathcal{S}$.

Algorithm 2. Signature Expansion

Require: An ontology $\mathcal{O}$ and a seed signature $seed\mathcal{S}$
Optional: An reasoner $\mathcal{R}$
Ensure: An expanded signature $\mathcal{S}$ is provided.

for all $e \in seed\mathcal{S}$ **do**
  $\mathcal{S} = \mathcal{S} \cup \{e\}$
  if e isA Class
    $\mathcal{S} = \mathcal{S} \cup getSubClasses(e)$
    $\mathcal{S} = \mathcal{S} \cup getSuperClasses(e)$
  end if
  if e isA Object Property
    $\mathcal{S} = \mathcal{S} \cup getPropertyDomains(e)$
    $\mathcal{S} = \mathcal{S} \cup getPropertyRanges(e)$
  end if
  if e isA Data Property
    $domains = getPropertyDomains(e)$
    $\mathcal{S} = \mathcal{S} \cup domains$
  end if
  if e isA Individual
    $\mathcal{S} = \mathcal{S} \cup getTypes(e)$
    $relIndvs = getRelatedIndividuals(e)$
    for all e' $\in relIndvs$ **do**
      $\mathcal{S} = \mathcal{S} \cup getTypes(e')$
    end for
  end if
end for
return $\mathcal{S}$

6 Evaluation

This chapter briefly describes the experimental efforts made to evaluate the envisaged solution. For that, a user study was conducted with the developed CQP prototype aiming:

(i) To evaluate the proposal in terms of usage in real case scenarios;

(ii) To compare with the results obtained in the preliminary experiments with the state-of-the-art approaches (cf. Chapter 3).

The remainder of this chapter is organized as follows:

- First, the design of the experiments is explained;
- Second, the experimentation results are described;
- Third, observations taken from the experiments are reported;
- Fourth, a comparison with the results obtained in the preliminary experiments is presented;
- Fifth, some conclusions are taken based on the experimentation results and a comparison with the preliminary experiments is performed.

6.1 Set-up

This section describes the experiment with the CQB tool as follows:

- First, the questions of the experiment are defined;
- Second, the study participants and the CQB tool are described;
- Third, the evaluation procedure is presented;.

6.1.1 Questions

The experimentation design is similar to the one described in the preliminary experiments (cf. section 3.1) thus allowing the comparison of the achieved results by different tools.

Accordingly, the user study consisted in answering four different (complex) questions using the CQB prototype. These questions were similar to the questions in the preliminary experiments, however the questions parameters where modified (cf. Table 14). For instance, in the first question of the preliminary experiments it was asked the population number of the Las Vegas city while in this experiment it is asked the population number of the city of Los Angeles. Accordingly, while the question is the same, the parameters, in this case the city, changes thus maintaining the same difficulty of the questions. Therefore, as in the preliminary experiments, the difficulty level of the question increases according to the number of the question. It was also ensured that the given questions could be answered using the CQB tool.

Table 14 – Questions description

No.	Question	Time limit
1	Which is the population of Los Angeles (United States of America city)?	3
2	Name twenty (20) cities with a population less than one hundred (100) inhabitants.	7
3	Name fifteen (15) persons whose birth place and death place was Houston (United States of America city).	10
4	Name ten (10) persons whose birth place and death place was Houston (United States of America city) and that were born after 1st January of 1920.	10

6.1.2 Participant and Tools

The same five users selected in the preliminary experiments (cf. section 3.1.2) were selected for testing the CQB tool.

The study participants were advised to disregard previous knowledge as much as possible. Finally, the users were asked to answer the questions individually.

As in the preliminary experiments where the tools were tweaked to access the DBPedia repository also the CQB tool was parameterized in order to access the DBPedia SPARQL Endpoint. Moreover, due to the characteristics of the DBPedia ontology (e.g. domain and ranges of most properties are not explicitly defined) it was decided not to use the Contextualization component in this evaluation.

6.1.3 Procedure

Accordingly, a similar procedure as the one described in the preliminary experiments was conducted.

Once again, the time limit for answering all the questions was 30 minutes. However, if a study participant was not able to answer a question within a certain time (as depicted in Table 14)

the user was advised to advance to the next question. Still, the user could answer the previous questions at any time.

Further, the experiments occurred in a lab so that the team could witness the efforts and the adopted approaches. No further restrictions were made.

In the end, the users were asked to answer a questionnaire. This questionnaire was similar to the one given in the preliminary experiments. On the one hand, the questionnaire served to compare the proposal with the state-of-the-art. On the other hand, the questionnaire served to evaluate the users' feedback on the *usefulness,* and *usability* of the proposal but also their *intention to use* such approach.

6.2 Results

This section demonstrates the obtained results in the CQB trial as follows:

- First, the answers to the questions in the experiment are detailed;
- Second, the answers on the questionnaire are analyzed.

6.2.1 Answers

As in the preliminary experiments (cf. section 3.2) the answers were evaluated according to two different dimensions:

(i) A Boolean evaluation (true or false) on if the users were able or not to answer a question;

(ii) The evaluation, in percentile, of the correctness of the given answers (i.e. measure the quality of the answers).

Table 15 demonstrates the analysis on the answered (✓) and not answered (×) questions.

Table 15 – Answered and not answered questions.

	Question 1	**Question 2**	**Question 3**	**Question 4**
User 1	✓	✓	✓	✓
User 2	✓	✓	✓	✓
User 3	✓	✓	✓	✓
User 4	✓	✓	✓	✓
User 5	✓	✓	✓	✓

As can be observed, all questions have been answered by all users. However, users not always correctly answered the questions. In fact, one of the users did not give a completely correct

answer to the fourth question. The other users correctly answered every question. Table 16 depicts the correctness of the answers provided by the users for each question.

Table 16 – Percentile of correct resources retrieved in each question.

	Question 1	Question 2	Question 3	Question 4
User 1	100%	100%	100%	60%
User 2	100%	100%	100%	100%
User 3	100%	100%	100%	100%
User 4	100%	100%	100%	100%
User 5	100%	100%	100%	100%

Figure 41 depicts the percentile of the overall correctness of the given answers grouped by question.

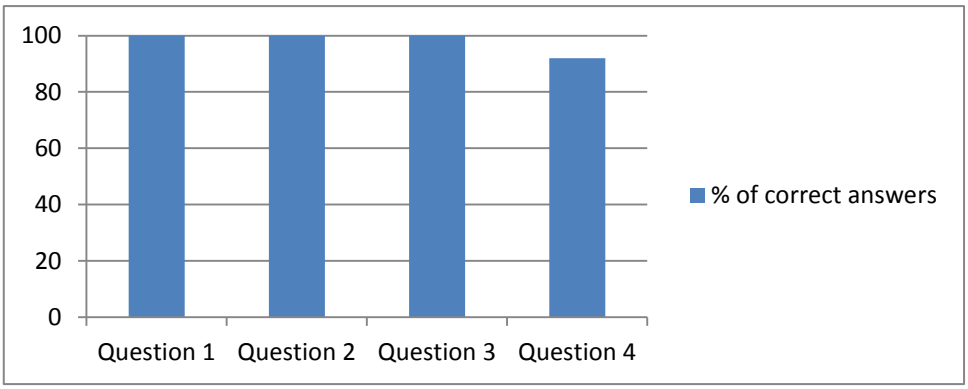

Figure 41 – Percentile of correct answers (grouped by question)

Moreover, Table 17 depicts the time spent by users to answer all the questions. According to it, most proficient users had finished the trial approximately 20 minutes earlier. Additionally, even the other users ended some time before the end of the time.

Table 17 – Time spent to answer all the questions.

User	Time (min)
User 1	10
User 2	10
User 3	15
User 4	20
User 5	22

6.2.2 Questionnaire

Results obtained from the questionnaire are described next.

The first question of the questionnaire asked the users if the CQB tool helped them to improve their performance while answering complex questions in comparison with any other systems. All users answered that the CQB tool effectively improved their performance while answering the proposed questions. Moreover, although it was needed some time to get confortable with the tool, it was easier to understand than the Virtuoso SPARQL Query Editor thus being better for non-SPARQL proficient users.

Further, the users said that the tool was easier to use. However, all stated that the supplied GUI could be improved in order to be more "user-friendly" and improve usability.

Furthermore, the questionnaire questioned the users about the encountered problems while answering the questions. Users that were not familiar with the concept of ontology had some difficulty while perceiving the differences between the ontological entities types (e.g. classes, object properties) and their role in the search/query.

Further on, most of the users stated that they were fully satisfied with the returned results.

Also, all the users stated that they would use this tool for question answering. Accordingly, users stated that the more increased the question' complexity the more the CQB tool seemed appropriate.

Moreover, when asked about any other tool they know that was better than the CQB tool for question answering none of the users proposed a different tool.

Finally, all users provided suggestions for improve the CQB' GUI. Accordingly, some users expressed that may be beneficial to join some of the existing tabs in order to simplify the GUI. Moreover, the edition of created constraints should be provided.

6.3 Observations

During the trial, the team observed how the users dealt with the CQB tool and how they managed to answer the questions. In this way, it was possible:

(i) To see what were the most significant problems that users encountered while using the tool;

(ii) To identify which functionalities the users used more often;

(iii) To verify how the users answered the questions (e.g. which information did they found).

While observing, the team perceived that users needed some time to understand how the tool work and how they should use the functionalities. Accordingly, they took some time to answer the first questions. However, after being familiar with the tool, users quickly answered the remainder questions. For instance, if one question was a refinement of a previously answered question (e.g. question 4 refines question 3), the users answered this new question very quickly. Moreover, when accustomed with the tool, non-proficient users' performance was close to the one of proficient users.

Moreover, surveyors noticed that users reached their answers by different paths. Accordingly, users correctly answered the questions although giving different answers. This is mainly due to the dynamic nature of the repository schema and the lack of maintenance over the repository knowledge thus allowing different representations of the same information. For instance, several relationships exist between "*SoccerPlayer*" and "*SoccerClub*" entities for representing the same information (e.g. "*teams*", "*clubs*", "*club*"). Additionally, when users did not find the wanted answers in a certain way, rapidly changed their search to try different ways thus highlighting the preponderant role of the user in the search task. Further, it emphasizes the iterative, incremental, and interactive properties of the proposal.

6.4 Analysis and Discussion

Comparing the results obtained from the CQB trial with the results obtained in the preliminary experiments it is possible to state that the proposal produced better results as shown in Figure 42. This figure depicts the number of given answers for each question in the preliminary experiments and in the proposal evaluation. On the one hand, it is visible that all users answered all questions using the CQB tool. On the other hand, it is shown that users did not give all the answers while using the State-of-the-Art tools.

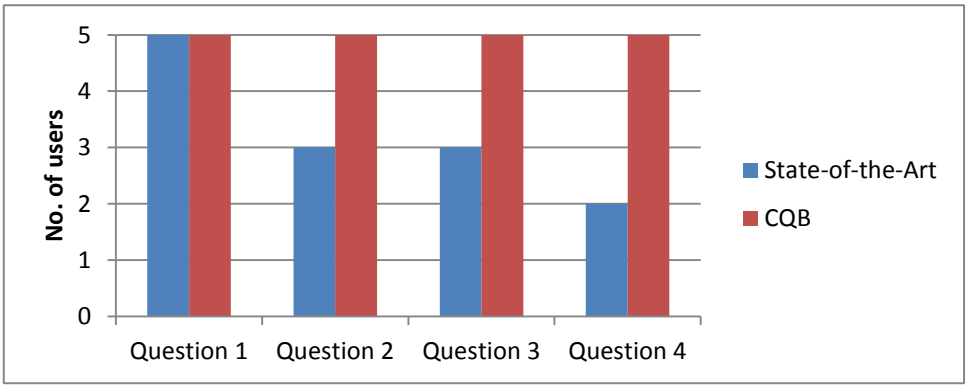

Figure 42 – Comparison on the number of given questions

Moreover, the results show that users answer the questions more correctly while using the CQB tool. Figure 43 depicts the overall correctness of the given answers (in percentile) for each question in the two trials. On the one hand, it is clear that the users were able to

correctly answer the proposed questions using the CQB tool. On the other hand, the correctness of the answers using the State-of-the-Art tools is much lower.

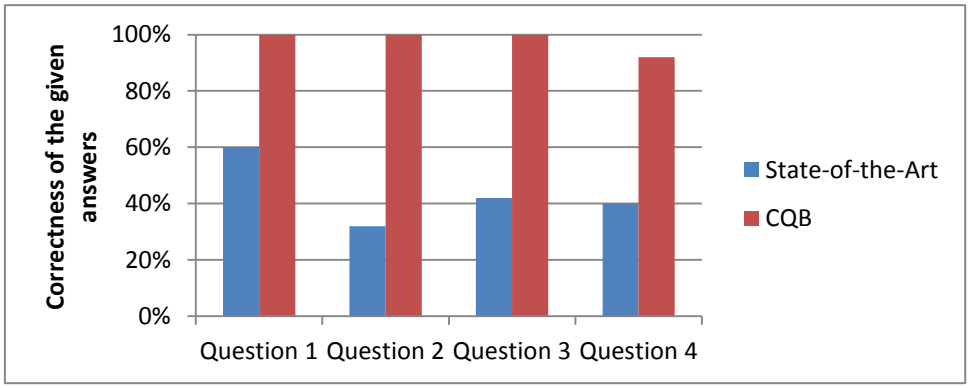

Figure 43 – Overall correctness of the given answers for each question

Therefore, it is possible to conclude that the CQB tool improves the information and search tasks by allowing users to find the necessary information to query the repository. Moreover, the CQB tool allows the creation of formal queries in a transparent way that correctly answers the proposed questions more effectively than the other tools.

Furthermore, for answering simple questions, both the CQB tool and Wikipedia Search achieve good results (e.g. the first question). However, the CQB tool lowers the probability of users making errors (cf. Figure 43). Nevertheless, when answering complex questions, users achieve better results with the CQB tool.

Additionally, it was visible that the CQB tool also boosts the users' performance while answering complex questions in two dimensions:

(i) The users give more complete and correct answers;

(ii) The users answer the questions in less time.

Yet, it was noticed that users needed some time to adapt to the CQB tool. Accordingly, it was evident that the other tools were easier to understand then the CQB tool. Still, after being comfortable with the tool, the users did not have any troubles when handling it. However, the users expressed concerns about the supplied GUI. For instance, users complained about the high number of tabs (seven). Accordingly, users suggested the aggregation of some similar functionalities provided in different tabs in only one tab. Moreover, it was suggested to hide the complexity over the ontology-related terms (e.g. classes, properties, individuals). In this respect, it is worth to recall that the GUI component was not the aim of this work (as previously stated). Accordingly, the GUI was barely developed to foster the evaluation of the proposed API. In this sense, it was expected that users would submit complaints about the supplied GUI.

6.5 Concluding Remarks

The results obtained in the performed trial shows that the CQB tool performs well while answering complex questions enabling the users to fully and correctly answer the proposed questions. Accordingly, the most surprisingly result of the experiment is that all users were able to answer the proposed questions only using the CQB tool. In this sense, it is evident that although the Contextualization component was not used the results were satisfying. However, it is intended to improve the Contextualization component in order to adapt to the different repositories ontologies' characteristics thus enhancing the developed prototype.

Finally, based on the experimentation results, it is possible to conclude that that:

- The more the users are proficient with a search approach (traditional text-based search vs. ontology-based search), the fast they answer the questions and less intellectual effort they put on the task;
- The time spent to answer a question with the CQB system decreased in the later questions. This is even more relevant considering that these questions were not simpler than the earlier questions.

7 Conclusion

The aims of this chapter are three-fold:

- First, the work achieved is succinctly described. Further, conclusions obtained from the experiments performed with the proposed application and a comparison with state-of-the-art is presented;
- Second, the thesis statement is validated;
- Third, it presents some ongoing and future work.

7.1 Summary

The work presented in this thesis aims to exceed the existing difficulties experienced by users while answering complex questions. In that sense, a system for answering complex questions based on knowledge stored in ontology-based repositories is proposed.

Preliminary experiments (cf. Chapter 3) on state-of-the-art applications (cf. section 2.3) revealed that such applications do not perform well while answering complex questions. For instance, users usually need to exploit several different applications in order to fully answer a complex question. Moreover, such applications usually lack some fundamental features that allows the user to perceive the repository's underlying ontology/schema (e.g. retrieve the common values of a given property). Accordingly, applications fail to bridge the gap between the user interpretation over some question and how the information to answer such question is captured in the repository. This experiment also revealed that only SPARQL proficient users were able to fully and correctly answer complex questions. However, even those might benefit from some mechanisms that allow them to search and navigate through the repository's ontology in order to acknowledge it and therefore be able to formulate the desired SPARQL queries. Accordingly, it was concluded that non-proficient users were not capable of answering complex questions. Moreover, although being capable of answering complex questions, proficient users had some difficulties in understanding the underlying ontologies.

Consequently, and according to the preliminary experiments, it was visible that users may benefit from some basic features that help them answer complex questions. For that reason, a proposal that relies on an API with six high-level components was proposed to provide a set of functionalities that allows one to navigate iteratively, incrementally and interactively through the repository knowledge base thus enabling the user to understand its data and the underlying schema (cf. Chapter 4). Finally, it provides the suitable mechanisms to properly formulate a formal query in order to query the repository and retrieve the results. This proposal was further prototyped (cf. Chapter 5).

The experiments with the CQB prototype (cf. Chapter 6) showed that the proposal effectively helps users answering complex questions. Accordingly, the experimentation results show that the CQB system boosts the users' performance while answering complex questions such that:

(i) The users answered all the proposed questions;

(ii) The users give more complete and correct answers;

(iii) The users answer the questions in less time.

Moreover, it was evident that users were able to easily, efficiently and effectively navigate through the repository using the CQB tool thereby building a vision on the repository's underlying schema. In this sense, it was also noticeable that the CQB tool allows users to find the necessary information to query the repository and create formal queries in a transparent way that correctly answers the proposed questions more successfully than the other tools. In this context, similar to the related work approaches, the proposal also emphasizes the user role in the search, browsing and navigation process. However, users were more satisfied with the control that they had using the CQB tool. In this respect, it is necessary to emphasize that users with the same expertise may achieve different solutions for the same complex question. Even though, some solutions might be better than others.

Further, the more the users are proficient with a search approach (text-based search vs. ontology-based search), the fast they answer the questions and less intellectual effort they put on the task. Observations showed that the time spent to answer a question with the CQB system decreased in the latter questions despite these questions were not simpler than the earlier ones. Moreover, users expressed their sympathy for the CQB approach, while answering the questions faster with this system. However, now and then the system returned an overwhelming number of results thus adding complexity to the search task which emphasizes the need of the contextualization component. Therefore, this issue is requiring our current and future attention.

Another identified major issue concerns the GUI module. In fact the users expressed concerns about the supplied GUI, suggesting the need to better track the results/iteration. In this respect, the GUI must automatically adapt/change based on several factors such as (i) the user proficiency, and (ii) the content' complexity of the provided semantic context (e.g. shown by means of a tree or a graph).

Additionally, the same experiments also revealed that users may benefit from the contextualization process because it permits the system to make more accurate suggestions and reduces the text-based search space.

7.2 Thesis Validation

Aligning the thesis statement with the proposed contributions, allows argument supporting the validity of the thesis statement. This thesis advocates:

- "*answering complex questions on semi-structured repositories is a task that ultimately depends on the users*":

The proposed approach consists on a user centric process to help user answering complex questions.

- "*to foster the task accomplishment users' supporting mechanisms are needed for helping them searching, navigating and querying iteratively, incrementally and interactively such semi-structured repositories*":

The proposed process allows users to iteratively, incrementally and interactively exploit a set of functionalities that support them to find answers for complex questions.

- "*in order to bridge the gap between the users' conceptualization underlying the question and the domain conceptualization expressed in the repository that is used to answer such question*":

Opposed to the users' conceptualization of the question, the proposed process enables users envisioning how information is actually structured and stored in repositories. Thus, it bridges the gap between different conceptualizations over the same domain. Yet, the proposed system supports the translation of the set of requirements posed by the user during her/his quest for information to a query language supported by the repository in order to properly query it and then retrieve the expected answers. Finally, as the experiments demonstrated, the proposed contributions allow users to improve their performance as well the obtained results when comparing to similar state of-the-art tools.

Accordingly, even if future work is necessary in this subject (e.g. having a better GUI), the proposed thesis is deemed valid.

7.3 Future Work

During the development of the prototype, some other ideas on how to help users in the complex querying task emerged. Accordingly, it was envisaged the integration of orthogonal ontological dimensions (e.g. time and space) during the information quest. In this sense, users could properly question the system over such dimensions. However, to add this component, some maintenance over the repository data should be performed. For instance, the

ontologies expressing the repository knowledge should be enriched with time and space information. Moreover, the adoption of thesaurus in the Object Mapping component is being studied in order to enhance the NLP techniques.

Furthermore, the most notable difference of this work to related work is the emphasis on focusing on the relevant search space (the context) and, therefore, discarding irrelevant information. According to this understanding, it was proposed the adoption and integration of modularization techniques into tasks/functions such as object mapping, entity information, relationship searcher and constraints specification. As a result, these functions become not completely deterministic because they rely on the concept of "context" to scale down their outcome. However, because of the characteristics of the DBPedia ontology, the obtained results were not satisfactory and therefore this will continue to deserve our attention. Accordingly, the developed module extraction algorithm should be improved and adapted to different ontologies characteristics.

Finally, users' feedback on the implemented prototype suggests the adoption of some new functionality. Accordingly, a component responsible for providing some statistical information that might help the users on their decisions and recommendations based on the users' past experiences are being studied. Moreover, in the near future, the GUI component should be enhanced with some semantic information according to the context of the query. For instance, the Contextualization component should provide an overview over the related entities to a certain context (e.g. exploiting ontology summarization algorithms). In this sense, the adoption of an iterative, incremental and interactive process as described in [Brandão, Maio, and Silva 2012b] is being studied. Additionally, some other GUI features should be developed.

References

d' Aquin, M., A. Schlicht, H. Stuckenschmidt, and M. Sabou. 2009. "Criteria and Evaluation for Ontology Modularization Techniques." In *Modular Ontologies*, ed. H. Stuckenschmidt, C. Parent, and S. Spaccapietra, 5445:67–89. Springer Berlin Heidelberg.

Baader, F., D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider. 2007. *The description logic handbook*. Cambridge university press.

Berners-Lee, T. 1989. *Information management: A proposal.*

Berners-Lee, T. 2008. Transcript: Sir Tim Berners-Lee Talks with Talis about the Semantic *Web.* February. http://talis-podcasts.s3.amazonaws.com/twt20080207_TimBL.html.

Berners-Lee, T. 2009. *Linked Data - Design Issues*. http://www.w3.org/DesignIssues/LinkedData.html.

Berners-Lee, T., and J. Hendler. 2001. *Nature Debates: Scientific publishing on the "semantic web.".*

Berners-Lee, T., J. Hendler, and O. Lassila. 2001. "The Semantic Web." In *Scientific American* (May): 29–37.

Bezerra, C., F. Freitas, J. Euzenat, A. Zimmermann, and G.I.D.A. Ireland. 2009. "An Approach for Ontology Modularization." In *Modular Ontologies*, ed. H. Stuckenschmidt, C. Parent, and S. Spaccapietra, 5445:67–89. Springer Berlin Heidelberg.

Bishop, B., A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. 2011. "OWLIM: A family of scalable semantic repositories." In *Semantic Web* 2 (1): 33–42.

Bizer, C., T. Heath, and T. Berners-Lee. 2009. "Linked data-the story so far." In *International Journal on Semantic Web and Information Systems (IJSWIS)* 5 (3): 1–22.

Boag, S., D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. 2010. XQuery 1.0: An XML Query Language (Second Edition). *XQuery 1.0: An XML Query Language (Second Edition). W3C Recommendation.* http://www.w3.org/TR/xquery/.

Borst, W.N. 1997. *Construction of engineering ontologies for knowledge sharing and reuse*. Universiteit Twente.

Brandão, R., P. Maio, and N. Silva. 2012a. "Enhancing LOD Complex Query Building with Context." In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT'12)*. Macau, China.

Brandão, R., P. Maio, and N. Silva. 2012b. "I3OM – An Iterative, Incremental and Interactive Approach for Ontology Navigation based on Ontology Modularization". In *4th International Conference on Knowledge Engineering and Ontology Development*. Barcelona, Spain.

Brickley, D., and R. V. Guha. 2000. Resource Description Framework (RDF) Schema Specification 1.0. *Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation.* March. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

Broekstra, J., A. Kampman, and F. Van Harmelen. 2002. "Sesame: A generic architecture for storing and querying rdf and rdf schema". In *The Semantic Web—ISWC 2002* 2342/2002. Lecture Notes in Computer Science: 54–68.

Chali, Y., S.R. Joty, and S.A. Hasan. 2009. "Complex question answering: unsupervised learning approaches and experiments." In *Journal of Artificial Intelligence Research* 35 (1): 1.

DBPedia. 2007. DBPedia. http://wiki.dbpedia.org.

Dzbor, M., E. Motta, C. Buil, J. Gomez, O. Görlitz, and H. Lewen. 2006. "Developing ontologies in OWL: An observational study." In *OWL: Experiences and Directions*.

Faceted Wikipedia Search. 2012. Faceted Wikipedia Search. http://dbpedia.neofonie.de/browse/.

Fensel, D., J. Hendler, H. Lieberman, and W. Wahlster. 2003. *Spinning the Semantic Web*. MIT Press.

Franconi, E., P. Guagliardo, and M. Trevisan. 2010. "An intelligent query interface based on ontology navigation." In *Workshop on Visual Interfaces to the Social and Semantic Web (VISSW2010)*. Vol. 10.

gFacet. 2012. gFacet - Visual Data Web. *gFacet - Visual Data Web*. http://www.visualdataweb.org/gfacet.php.

Giunchiglia, F., U. Kharkevich, and I. Zaihrayeu. 2009. "Concept search." In *The Semantic Web: Research and Applications* 5554/2009. Lecture Notes in Computer Science: 429–444.

Goker, A., and J. Davies. 2009. *Information retrieval: searching in the 21st century*. Wiley.

Gruber, T.R. 1993. "A translation approach to portable ontology specifications." In *Knowledge acquisition* 5 (2): 199–220.

Gruber, T.R. 1995. "Toward principles for the design of ontologies used for knowledge sharing." In *International journal of human computer studies* 43 (5): 907–928.

Hahn, R., C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. 2010. "Faceted Wikipedia Search." In *13th International Conference on Business Information Systems (BIS 2010)*, 47:1–11. Springer.

Hearst, M. 2006. "Design recommendations for hierarchical faceted search interfaces." In *ACM SIGIR Workshop on Faceted Search*, 1–5.

Heim, P., S. Lohmann, and T. Stegemann. 2010. "Interactive Relationship Discovery via the Semantic Web." In *The Semantic Web: Research and Applications:* 6088/2010. 303–317.

Heim, P., J. Ziegler, and S. Lohmann. 2008. "gFacet: A Browser for the Web of Data." In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, 417:49–58.

Heim, P., and J. Ziegler. 2011. "Faceted Visual Exploration of Semantic Data." In *Human Aspects of Visualization*, ed. Achim Ebert, Alan Dix, Nahum Gershon, and Margit Pohl, 6431/2011:58–75. Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Hepp, M., P. De Leenheer, and A. de Moor. 2007. *Ontology management: Semantic Web, Semantic Web Services, and Business Applications*. Springer New York.

Hussain, S., and S.S.R. Abidi. 2010. "Extracting and merging contextualized ontology modules." In *Modular Ontologies - Proceedings of the Fifth International Workshop (WoMO 2011)*, 25–40. IOS Press.

Kolomiyets, O., and M.F. Moens. 2011. "A survey on question answering technology from an information retrieval perspective." In *Information Sciences* 181 (24): 5412–5434.

Lassila, O., and R. R. Swick. 1999. Resource Description Framework (RDF): Model and Syntax Specification. *Resource Description Framework (RDF): Model and Syntax Specification. W3C Recommendation.* February. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

Li, N., E. Motta, and M. d' Aquin. 2010. "Ontology summarization: an analysis and an evaluation." In *International Workshop on Evaluation of Semantic Technologies* (IWEST'2008). Shangai, China, November 8.

Linked Data. 2012. Linked Data | Linked Data - Connect Distributed Data across the Web. http://linkeddata.org/.

Lucene. 2012. Apache Lucene - Apache Lucene Core. http://lucene.apache.org/core/.

Microsoft-Corporation. 2008. Enterprise Search: For FAST Customers. http://www.microsoft.com/enterprisesearch/en/us/fast-customer.aspx.

Morville, P., and J. Callender. 2010. *Search patterns*. O'Reilly Media, Inc.

Motta, E., P. Mulholland, S. Peroni, M. d' Aquin, J. Gomez-Perez, V. Mendez, and F. Zablith. 2011. "A novel approach to visualizing and navigating ontologies." In *The Semantic Web–ISWC 2011*: 470–486.

Ong, C.S., M.Y. Day, K.T. Chen, and W.L. Hsu. 2008. "User-centered evaluation of question answering systems." In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics* (ISI'2008), 286–287. IEEE.

van Ossenbruggen, J., L. Hardman, and L. Rutledge. 2006. "Hypermedia and the semantic web: a research agenda." In *Journal of Digital information* 3 (1).

OWL API. 2009. OWL API. *OWL API*. http://owlapi.sourceforge.net/.

Parent, C., and S. Spaccapietra. 2009. "An Overview of Modularity." In *Modular Ontologies*, ed. H. Stuckenschmidt, C. Parent, and S. Spaccapietra, 5445:5–23. Springer Berlin Heidelberg.

Prud'hommeaux, E., and A. Seaborne. 2008. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.

RelFinder. 2012. RelFinder. http://www.visualdataweb.org/relfinder/relfinder.php.

Smith, M., C. Welty, and D. McGuinness (eds.). 2004. *OWL Web Ontology Language Guide*. Recommendation. February 10. http://www.w3.org/TR/owl-guide/.

Stuckenschmidt, H., and M. Klein. 2004. "Structure-based partitioning of large concept hierarchies." In *The Semantic Web–ISWC 2004* 3298/2004: 289–303.

Stuckenschmidt, H., and A. Schlicht. 2009. "Structure-Based Partitioning of Large Ontologies." In *Modular Ontologies*, ed. H. Stuckenschmidt, C. Parent, and S. Spaccapietra, 5445:187–210. Springer Berlin Heidelberg.

Studer, R., V.R. Benjamins, and D. Fensel. 1998. "Knowledge engineering: principles and methods." In *Data & knowledge engineering* 25 (1): 161–197.

Teevan, J., S. Dumais, and Z. Gutt. 2008. "Challenges for supporting faceted search in large, heterogeneous corpora like the Web." In *Proceedings of HCIR 2008*: 6–8.

Tunkelang, D. 2009. "Faceted Search." In *Synthesis Lectures on Information Concepts, Retrieval, and Services* 1 (1) (January): 1–80.

Del Vescovo, C., B. Parsia, U. Sattler, and T. Schneider. 2011. "The modular structure of an ontology: atomic decomposition." In *Modular Ontologies - Proceedings of the Fifth International Workshop (WoMO 2011)*. IOS Press.

Virtuoso SPARQL Query Editor. 2012. Virtuoso SPARQL Query Editor. http://dbpedia.org/sparql.

Völkel, M., M. Krötzsch, D. Vrandecic, H. Haller, and R. Studer. 2006. "Semantic Wikipedia". In *European Semantic Web Conference 2006*, 585–594. ACM.

W3C. 2007. Semantic Web, and Other Technologies to Watch: January 2007 (24). http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24).

W3C. 2012a. World Wide Web Consortium (W3C). http://www.w3.org/.

W3C. 2012b. Rule Interchange Format (RIF) Current Status - W3C. http://www.w3.org/standards/techs/rif#w3c_all.

Warren, P. 2006. "Knowledge management and the semantic web: From scenario to technology." In *Intelligent Systems, IEEE* 21 (1): 53–59.

Watson. 2012. Watson Semantic Web Search. http://kmi-web05.open.ac.uk/WatsonWUI/.

Wikipedia. 2012. Wikipedia. http://www.wikipedia.org/.

Wolfram Alpha. 2012. Wolfram|Alpha: Computational Knowledge Engine. http://www.wolframalpha.com/.

World Search. 2010. World Search. http://www.microsoft.com/portugal/mldc/worldsearch/en/.

Yago2 spotlx. 2012. Yago 2 spotlx: A Core of Semantic Knowledge. https://d5gate.ag5.mpi-sb.mpg.de/webyagospotlx/WebInterface.
Zhang, X., G. Cheng, W.Y. Ge, and Y.Z. Qu. 2009. “Summarizing vocabularies in the global semantic web.” In *Journal of Computer Science and Technology* 24 (1): 165–174.