Enhancing LOD Complex Query Building with Context

Ricardo Brandão, Paulo Maio and Nuno Silva
Knowledge Engineering and Decision Support Research Center
School of Engineering, Polytechnic of Porto
Porto, Portugal
{jrmjb, pam, nps}@isep.ipp.pt

***Abstract*— Open ontology-described repositories are becoming very common in the web and in enterprises. These repositories are well-suited to answer complex queries, but in order to fully exploit their potential, the queries should be written in a user-demand basis, and not in a traditional static approach by software developers. Hence, the users are required (i) to know the underlying ontology(ies) and/to (ii) write formal queries. Yet, the users often lack such requirements. In this paper we first describe the observations made during manual complex querying process and present a systematization of the users' support wish list for building complex queries. Based on this systematization we propose an extended set of functionalities for a user-supporting system. Finally, we demonstrate their application in a walk-through example and their implementation within a prototype.**

Keywords - ontology; complex questions; knowledge management;

I. INTRODUCTION

Open information repositories described by means of ontologies are becoming very common both in the web (e.g. DBPedia) and enterprises (e.g. World Search). The World Search [1] project aims to create a system that supports healthcare professionals such as nurses, geriatricians and physiatrists finding information that answer complex questions evolving many concepts (not always explicitly stated in the ontology/schema), relations and repositories.

These repositories are often referred as Linked-Open Data repositories, or simply LOD. Due to its well-formed structure (and sometimes semantics), a new set of applications and demands are rising. However, (i) the schemes of these repositories are very dynamic, (ii) the requirements and semantics putted upon the schemas and upon the data are heterogeneous, dynamic and potentially unlimited, hence rising operational issues.

In this paper, we focus on supporting fully exploitation of these repositories, such that the questions are written in a user-demand basis, and not made available by the developer. In particular, we intend to support the users in building complex query, as they often lack fundamental required skills: (i) to know the underlying schemes/ontologies and (ii) to specify the queries formally (e.g. by means of query language).

This paper starts by reporting the team's investigations regarding the ways users make use of these repositories when trying to respond to complex questions, and (ii) how they expect to be supported on that task (section II). Observations showed that the repositories' front-ends do not provide enough support for the task. Based on these findings, we propose a set of functionalities that should be included in a complex-query supporting front-end (section III). A walk-through example illustrates the application of these functionalities (section IV). Section V describes the architecture and implementation decisions in the context of a larger effort to build a healthcare-related semantic repository. Finally, we draw some final remarks and describe the follow-up research directions (section VI).

II. INVESTIGATION

Because the domain knowledge of the World Search project is not mainstream and therefore often incomprehensible for non-experts, we decided to investigate the complex query building task with a common-sense question from the DBPedia [2]: "*All soccer players, who played as goalkeeper for a club that has a stadium with more than 40.000 seats and who are born in a country with more than 10 million inhabitants*". Obviously this question is not answerable through "search, browsing & navigation" operations without further data processing.

A possible UML representation of the underlying ontology of the question is depicted in Fig. 1. Notice that all the concepts, relations, attributes and values present in the aforementioned question are included in the ontology.

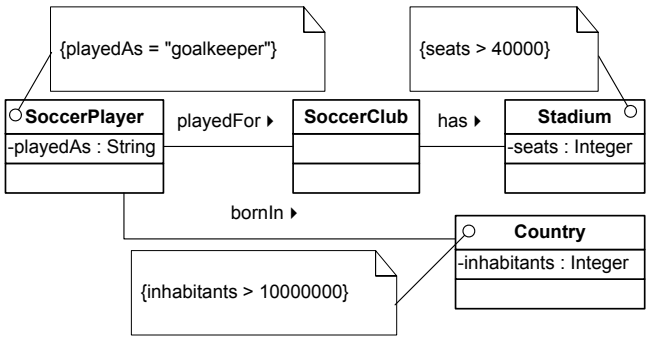

Figure 1. UML representation of the ontology underlying the question.

According to the DBPedia, the ontology module capable to answer this question is slightly different (Fig. 2).

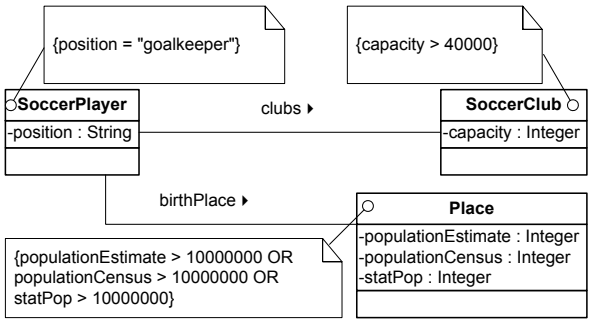

Figure 2. UML representation of the DBPedia proposed query to answer the mentioned question.

In order to evaluate current search tools, we conducted a trial with 6 users. The participants, mainly informatics students were very familiar with the usage of the general purpose search engines (e.g. Google, Bing) and faceted search. Three of them were familiar with the concept of ontology/schema, triples, LOD and SPARQL. None of them were familiar with the DBPedia repository and its schema/ontology.

We asked the users to answer the question making use of:

- Faceted Wikipedia Search [3], allows users to ask complex queries against Wikipedia based on the schema/ontology and information extracted from many different Wikipedia articles;
- RelFinder [4], a user-centered tool for interactively discover relationships between elements in the Semantic Web;
- gFacet [5] which is a front-end that allows formulating complex queries supporting the navigation through the ontologies;
- any other tool the participants wants.

Further, two restrictions were stated: (i) the maximum duration (75 minutes) and (ii) the location (in the lab) so that the team could witness the efforts and the adopted approaches. In the end, no participant succeeded.

Based on this result the team re-ran the experiment but this time the examiners supported the participants with some guidance and explanations. After the task completion the participants were inquired about their wish list for supporting complex query formulation. The lists were summarized in the following:

- Help find the attributes names whose values include a specific value;
- Provide abstractions for class expressions (e.g. $+10MPopulatedCountry = Country \cap \exists.populationtotal(> 10M)$
- Abstract the subsumption relation (e.g. the Country→PopulatedPlace→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);
- Combine all these requirements into a single tool.

III. PROPOSAL

In this section we describe our proposal independently from any concrete implementation or application domain. The proposal relies on a set of six high-level components that are suitable combined into a user application. 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. These components are summarized as follows.

A. Ontology-based Repository

The Ontology-based Repository component is where the data/information/knowledge necessary to answer the user complex questions is maintained. Thus, an ontology-based repository can be seen as an ontology. Ontologies may be expressed in a variety of ontology languages (e.g. OWL [6]) that define the ontology entities. Fortunately, most of these languages share the same kind of entities, often with different names but comparable interpretations. Thus, 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 [7].

B. 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. In that sense, it is envisaged the adoption and combination of three modularization techniques [8]: (i) ontology partitioning, (ii) module extraction and (iii) ontology summarization. Next, each one of these techniques is described according to their goals.

An ontology partitioning technique identifies the key topics of an ontology and splits it into several fragments [9].

Typically, each key topic gives rise to a fragment which is usually called as module (cf. Definition 2).

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 Nelements (modules) such that $\mathcal{P} = \{\mathcal{O}_1, \mathcal{O}_2, \dots, \mathcal{O}_N\}$.

A module extraction technique aims to extract a focused fragment (or module) of the original ontology given a specific topic of interest [10]. The topic of interest is captured by the notion of signature (cf. Definition 4).

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 $\mathcal{S}$.

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 [11], [12].

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:

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

C. Object Mapping

The Object Mapping component is responsible for mapping a natural language text introduced by the user to ontological entities. For that, it is envisaged a three steps approach as follows.

The first step is completely automatic. It consists in applying a set of Natural Language Processing (NLP) techniques (e.g. tokenization, stops words, lemmatization, stemming) to identify the relevant terms in a natural language text. This step is formalized as follows.

Definition 8 (Terms Identification) – The terms identification task is seen as a function $nlp: text \rightarrow \mathcal{T}$ 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 $text$.

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. 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 $e \in \mathcal{O}$ 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).

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). 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.

D. Relationship Searcher

The Relationship Searcher component is responsible for finding the ontological relations existing between two ontological entities in a given context.

Definition 10 (Finding Relationships) – The task of finding relationships between two ontological entities is a function $relSearch(e, e', \mathcal{O}, \mathcal{C}, length, dir) \rightarrow Rels$ 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 $(backward)$. ;
- $Rels$ 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 relationships 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 $n < length$ and $v \in]0, \infty[$ is a value expressing the relevance of the relationships path.

E. 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 in determining which logical operators (e.g. equal, greater than, contains, existential and universal quantifiers) 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. This process is formalized as follows.

Definition 11 (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 $\mathcal{C}$.

The second process consists in 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 12 (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.

The third process concerns the ability to construct the appropriate constraint given (i) a property, (ii) an operator $(op \in OP)$ and (iii) a property value (pv'). This process is formalized as follows.

Definition 13 (Constraint) – The task of constructing a constraint on a property is a function $constraint(p, \upsilon, pv') \rightarrow \lambda$ such that:

- $p \in \mathcal{O}$ is the ontological property on which the constraint applies on;
- $\upsilon \in \Upsilon$ is the logical operator applied in the constraint;
- pv' is a property value;
- λ is the resulting constraint on p such that υ is true for pv'.

F. 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[7]) 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.

Definition 14 (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.

IV. WALK-THROUGH EXAMPLE

To demonstrate our approach and how the aforementioned components might be exploited together by an application assisting the user to accomplish the task of asking complex questions, we present now a walk-through example. In this example, the user makes use of an application whose underlying repository is the DBPedia (i.e. $\mathcal{O}_{DBPedia}$) to find an appropriate answer to the previously mentioned question: "*All soccer players, who played as goalkeeper for a club that has a stadium with more than 40.000 seats and who are born in a country with more than 10 million inhabitants*".

Considering the question in hands, let us assume the user starts searching using the text "*soccer player*". Thus, the Object Mapping component is required to map such text to some ontology entities. The result of the first step is as follows:

$$nlp(\text{"soccer player"}) \rightarrow \mathcal{T}_1 : \mathcal{T}_1 = \{soccer, player\}$$

Hence, in the second step two 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. A possible result of function $map(\mathcal{T}_1, \mathcal{O}_{DBPedia}, \mathcal{O}_{DBPedia})$ is depicted in TABLE I where the confidence value of each pair term-entity is omitted by clarity reasons.

TABLE I. THE RESULTS OF THE *map* FUNCTION

Term	List of Entities
soccer	{SoccerManager, SoccerClub, SoccerPlayer, SoccerLeague}
player	{RugbyPlayer, GolfPlayer, SnookerPlayer, ChessPlayer, AmericanFootballPlayer, SoccerPlayer, BaseballPlayer, VolleyballPlayer, BadmintonPlayer, playerInTeam, GaelicGamesPlayer, GridironFootballPlayer, PokerPlayer, IceHockeyPlayer, TennisPlayer, CanadianFootballPlayer, BasketballPlayer}

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. Regarding the question in hands, consider that the user confirms the system suggestion: "*SoccerPlayer*". This entity becomes the current context.

Next, the system may contextualize the "*SoccerPlayer*" entity. For that, the system can extract a module of the ontology focused on this entity such that $\sigma(\mathcal{O}_{DBPedia}, \{SoccerPlayer\}) \rightarrow \mathcal{M}_1$. Yet, considering that the resulting module ($\mathcal{M}_1$) contains several entities that cannot be all (easily) represented in the GUI, the system may opt by summarize it such that $\varphi(\mathcal{M}_1) \rightarrow \mathcal{D}_1$. In that sense, let us admit that among other entities $\mathcal{D}_1$ contains the concepts "*SoccerClub*", "*Person*" and "*PopulatedPlace*" and none relationship between these concepts and "*SoccerPlayer*".

Afterward, as a second interaction, the user could request the system to find relationships between "*SoccerPlayer*" and "*SoccerClub*". The result of executing $relSearch(SoccerPlayer, SoccerClub, \mathcal{O}_{DBPedia}, \mathcal{M}_1, 1, forward)$ is $Rels_1$ which is partial depicted in TABLE II.

TABLE II. THE FIVE MOST RELEVANT RELATIONSHIPS BETWEEN CONCEPTS SOCCERPLAYER AND SOCCERCLUB

r	v
{SoccerPlayer, team, SoccerClub}	270601
{SoccerPlayer, clubs, SoccerClub}	243588
{SoccerPlayer, currentclub, SoccerClub}	36463
{SoccerPlayer, youthclubs, SoccerClub}	26396
{SoccerPlayer, title, SoccerClub}	108

It worth to notice that the property "*clubs*" required by the solution proposed in the DBPedia (cf. Fig. 2) is identified and suggested to the user. Since the user does not know that, let us admit s/he selects the property "*team*" instead "*clubs*".

Further, the user may repeat the process of finding relationships for the concepts "*SoccerPlayer*" and "*PopulatedPlace*". The consequent result is partial depicted in TABLE III.

TABLE III. THE FIVE MOST RELEVANT RELATIONSHIPS BETWEEN CONCEPT SOCCERPLAYER AND POPULATEDPLACE

r	v
{SoccerPlayer, birthPlace, PopulatedPlace}	104931
{SoccerPlayer, placeOfBirth, PopulatedPlace }	98410
{SoccerPlayer, countryofbirth, PopulatedPlace }	35649
{SoccerPlayer, cityofbirth, PopulatedPlace }	30979
{SoccerPlayer, deathPlace PopulatedPlace }	5010

In this case, consider that the user selects the property "*countryofbirth*" because it is more meaningful than the others by the fact of containing the word "country", which is used in the question in hands. The specification made by the user after the just described three interactions is graphically depicted in Fig. 3.

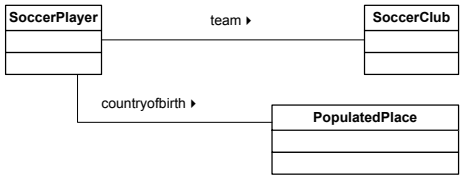

Figure 3. The specification made by the user after three interactions.

Along with the question specification, the system might provide the user with some results. Hence, consider $\mathcal{G}_1$ as the connected graph represented in Fig. 3 and Λ and Z as empty sets. Therefore, the output of $query_{SPARQL}(\mathcal{G}_1, \emptyset, \emptyset)$ is the query represented in Fig. 4.

```
SELECT ?player, ?club, ?place
{
   ?player a <http://dbpedia.org/ontology/SoccerPlayer> .
   ?club a <http://dbpedia.org/ontology/SoccerClub> .
   ?place a <http://dbpedia.org/ontology/PopulatedPlace> .
   ?player <http://dbpedia.org/ontology/team> ?club .
   ?player <http://dbpedia.org/property/countryofbirth> ?place .
}
```

Figure 4. The query SPARQL resulting from $\mathcal{G}_1$

By executing this query the system would return approximately 34523 results. In face of the huge amount of results returned, the user decides to start filtering them. For

that, the user may request a search for "*goalkeeper*" in the context of the entity "*SoccerPlayer*". Again, the Object Mapping component is exploited. As a result, the system would perceive that the term "*goalkeeper*" is mostly found as a (part of a) value of the data property "*position*". By receiving this suggestion, the user may select such property and request a new constraint for it. As that, the system runs sequentially two processes of the Constraints Specification component: (i) the Applicable Operators and (ii) the Property Common Values. A set of possible results of these processes are present in TABLE IV and TABLE V respectively.

TABLE IV. THE APPLICABLE OPERATORS TO THE PROPERTY "POSITION"

Property	Range	Applicable Operators
position	string	{=, ≠, <, >, ≤, ≥, contains, starts, ends, }

TABLE V. THE FIVE MOST COMMON VALUES OF PROPERTY "POSITION"

Property (p)	Property Value (pv')	v
position	Midfielder	14135
	Defender	11157
	Striker	8179
	Goalkeeper	5889
	Forward	4658

By inspecting the most common values the "*position*" property, the user may request to create a constraint such that $constraint("position", =, "goalkeeper")$. As a result, a new query is executed against the repository, returning approximately 3178 results.

For the required restriction on concept "*PopulatedPlace*" the user may adopt a similar process to the one described for "*goalkeeper*". By doing that using terms such as "*inhabitants*" or "*habitants*" the user would not be able to find any relevant and suitable property. This fact leads the user to search alternative terms such as synonyms[1]. By using the synonym "*population*", properties such as (i) "*populationTotal*", (ii) "*totalPopulation*" and (iii) "*populationEstimate*" will be suggested. User attempts with the two first properties would have as outcome an empty set of results. Through the third attempt (i.e. "*populationEstimate*") the set of results would have approximately 1462 elements.

To restrict the results to those players related to clubs whose stadiums have more than 40.000 seats, the user may search for the term "*stadium*" which will be disambiguated to the ontological concept "*Stadium*". Further, the user requests the system to find relationships between "*SoccerClub*" and "*Stadium*". The most relevant property between these entities is "*ground*". Further, searching for the term "*seats*" in the context of "*Stadium*" the system will suggest the property "*seatingCapacity*" as one of the most relevant. Consequently, the user specifies the respective constraint. The entirely user specification is graphically depicted in Fig. 5.

[1] The synonyms must also be suggested/requested by/to the system.

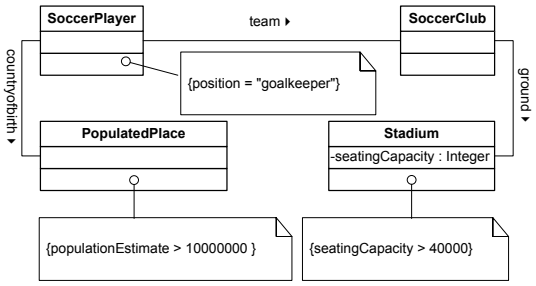

Figure 5. The final specification made by the user.

By executing the query resulting from this specification, the user obtains approximately 406 results that definitely answer the question in hands.

V. IMPLEMENTATION

This section describes our implementation efforts of the proposed ideas in the context of the World Search project.

A. World Search Architecture

The World Search system architecture is illustrated in Fig. 6.

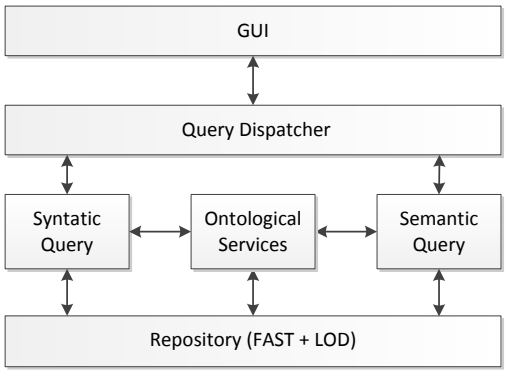

Figure 6. The architecture of the WordSearch project.

This architecture has six main modules with the following responsibilities:

- 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 [13] for indexing purposes and (ii) a data source meeting the Linking Open Data principles [14].

B. Developed Prototype

The components of our proposal integrate into the World Search architecture as illustrated in Fig. 7.

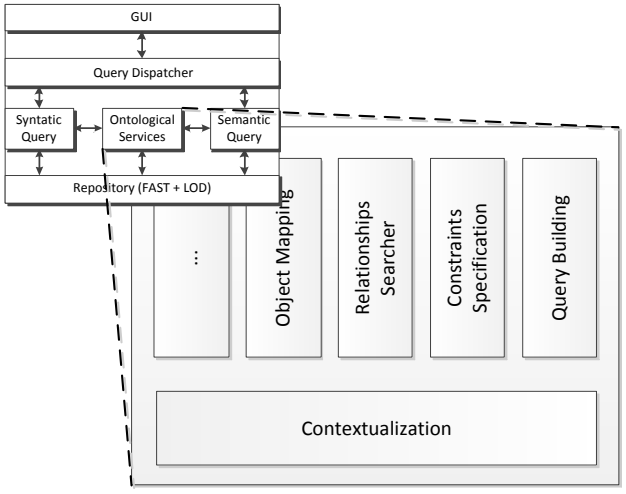

Figure 7. The proposed components integrated in the World Search architecture.

1) Contextualization

The Contextualization component was first deployed making use of the following third-party modularization tools:

- The Ontology Partitioning algorithm provided in the Swoop tool [15];
- The Module Extraction algorithm proposed in [16];
- The KCE Ontology Summarization algorithm [17].

All these algorithms were selected because (i) of their relevance in the state-of-the-art and (ii) they are publicly available. However, early experiments (mostly in the domain of health care) show that these algorithms do not perform well and, therefore, lead to poor results. The main reasons for that rely on the characteristics of the adopted ontologies, which have a reduced expressivity: some of them are simple hierarchical taxonomies and another uses opaque entities' names. To overcome some of the identified problems, these algorithms are being enhanced through the adoption of some basic heuristics and structural methods. Additionally, information extracted from individuals dimension (i.e. $\mathcal{A}$-boxes) analysis is also used to feed these algorithms.

2) Object Mapping

The suggestions provided by the Object Mapping component are retrieved by an indexing mechanism (FAST). This mechanism indexes the content of a configurable set of selected properties determined by the entity type. Current configuration is represented in TABLE VI. Each selected property is associated to a weight-value, which is used for ranking purposes.

TABLE VI. CURRENT INDEXED SELECTED PROPERTIES.

Entity Type	Dimensions			
	$\mathcal{T}$-box		$\mathcal{A}$-box	
	Property	Weight	Property	Weight
Class	rdf[2]: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 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.

3) Relationship Searcher

The Relationship Searcher component implementation relies on a set of a pre-defined set of SPARQL queries against the repository. 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.

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. This interpretation function is able to dynamically exploit the XML Built-in Datatype hierarchy [18] to infer the best possible mapping. For non-recognized ranges, the function exploits a static list of mappings that can be seeded by users with especial rights.

The most common values of a given property are determined through a set of parameterized SPARQL queries against the repository. These queries take into consideration the intended context on which the property is being used.

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.

Finally, 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. This status is

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

used namely to keep tracking the ontology entities the user is interesting in and the current context.

VI. Discussion

This section provides a discussion upon the proposed contributions both in comparison with the related work and introspectively.

A. Related Work

In [4] the authors present the RelFinder tool based on the ORVI four step process: Object Mapping, Relationship Search, Visualization, and Interactive Exploration. This tool comprehends the Object Mapping and Relationship Searcher components only. Unlike our proposal, the retrieved results do not take into consideration the context. Further, while this tool supports the user in the process of discovering relationships between ontological entities, our approach combines the two components supporting the user building the complex queries.

In [3] the authors propose a method and a tool that allows/supports users to access and explore Semantic Web using faceted search, narrowing down large sets of data. The underlying "graph provides a coherent representation" of the (search) facets thus enhancing the relationships between such facets. The authors propose a three-stage model consisting of (1) Goal formation, where the user typically specifies one or more concepts to determine the initial search space, then (2) the user incrementally constructs the search space by exploiting the relationships between the starting concepts and others linked to them, and finally (3) the Multi-perspective exploration and sense-making stage where the user explores the space and "make sense of the information and relations contained in it". The three-stage model has been implemented in gFacet [5] tool, that visually represents the facets in a graph. However, this tool does not allow the intersection of multiple filters (e.g. "soccer players born in a country with more than 10 million inhabitants"). Moreover, faceted search has some limitations when dealing with vast number of entities. Our approach deals with this issue through contextualization.

B. Conclusion and Future Work

The most notable difference of our proposal to related work is the emphasis on focusing on the relevant search space (the context) and, therefore, discarding irrelevant information. According to this understanding, we proposed the adoption and integration of modularization techniques into tasks/functions such as object mapping, 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. In this respect, preliminary experiments highlighted the need to considerably improve state-of-the-art modularization algorithms. This will deserve our future attention. On the other hand, the same experiments also reveal that users benefit from the contextualization process because it permits the system to make more accurate suggestions and reduces text-based searches made by the user.

Similar to the related work approaches, our proposal also emphasizes the user role in the search, browsing and navigation process. In this respect, it is necessary to emphasize that users with the same expertise may achieve different solutions for the same complex question. This is easily perceived by comparing the solution achieved in the walk-through example with the proposed DBPedia solution. Even though, some solutions might be better than others.

Based on the feedback received from users that are already using the developed prototype, we are planning to evolve the prototype with a component responsible for providing some statistical information that might help the user on their decisions and recommendations based on the users' past experiences.

Acknowledgment

This work is partially supported by the Portuguese projects: World Search (QREN11495) and OOBIAN (QREN 12677), both funded by FEDER through the COMPETE program for operational factors of competitivity.

References

[1] 'World Search', 2010: http://www.microsoft.com/portugal/mldc/worldsearch/en/.
[2] 'DBPedia', 2007: http://wiki.dbpedia.org.
[3] P. Heim and J. Ziegler, 'Faceted visual exploration of semantic data', Human Aspects of Visualization, vol. 6431/2011, pp. 58–75, 2011.
[4] P. Heim, S. Lohmann, and T. Stegemann, 'Interactive relationship discovery via the semantic web', The Semantic Web: Research and Applications, pp. 303–317, 2010.
[5] 'gFacet'. http://www.visualdataweb.org/gfacet/gfacet.php.
[6] M. Smith, C. Welty, and D. M. (eds.), 'OWL Web Ontology Language Guide', Recommendation, Feb. 2004.
[7] E. Prud'hommeaux and A. Seaborne, 'SPARQL Query Language for RDF', 2008. http://www.w3.org/TR/rdf-sparql-query/.
[8] C. Parent and S. Spaccapietra, 'An Overview of Modularity', in Modular Ontologies, vol. 5445, H. Stuckenschmidt, C. Parent, and S. Spaccapietra, Springer, 2009, pp. 5–23.
[9] H. Stuckenschmidt and A. Schlicht, 'Structure-Based Partitioning of Large Ontologies', in Modular Ontologies, vol. 5445; Springer, 2009.
[10] S. Hussain and S. S. R. Abidi, 'Extracting and merging contextualized ontology modules', The Fourth International Workshop on Modular Ontologies, 2010, pp. 25–40.
[11] X. Zhang, G. Cheng, W.-Y. Ge, and Y.-Z. Qu, 'Summarizing Vocabularies in the Global Semantic Web', Journal of Computer Science and Technology, vol. 24, no. 1, pp. 165–174, 2009.
[12] N. Li, E. Motta, and M. d' Aquin, 'Ontology summarization: an analysis and an evaluation', International Workshop on Evaluation of Semantic Technologies (IWEST 2010), Shangai, China, 2010.
[13] MSFT, 'Enterprise Search: For FAST Customers', 2008. http://www.microsoft.com/enterprisesearch/en/us/fast-customer.aspx.
[14] T. Berners-Lee, 'Linked Data - Design Issues', 2009. http://www.w3.org/DesignIssues/LinkedData.html.
[15] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. Hendler, 'Swoop: A web ontology editing browser', Web Semantics: Science, Services and Agents on the WWW, vol. 4, no. 2, pp. 144–153, 2006.
[16] P. Doran, V. Tamma, and L. Iannone, 'Ontology module extraction for ontology reuse: an ontology engineering perspective', The sixteenth ACM conference on information and knowledge management, New York, NY, USA, 2007, pp. 61–70.
[17] S. Peroni, E. Motta, and M. d' Aquin, 'Identifying key concepts in an ontology, through the integration of cognitive principles with statistical and topological measures', The Semantic Web, 2008.
[18] P. Biron and A. Malhotra, 'XML Schema Part 2: Datatypes Second Edition'. [Online]. Available: http://www.w3.org/TR/xmlschema-2/.