International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:3, 2015

A Novel Framework for User-Friendly Ontology-Mediated Access to Relational Databases

Efthymios Chondrogiannis, Vassiliki Andronikou, Efstathios Karanastasis, Theodora Varvarigou

***Abstract*—**A large amount of data is typically stored in relational databases (DB). The latter can efficiently handle user queries which intend to elicit the appropriate information from data sources. However, direct access and use of this data requires the end users to have an adequate technical background, while they should also cope with the internal data structure and values presented. Consequently the information retrieval is a quite difficult process even for IT or DB experts, taking into account the limited contributions of relational databases from the conceptual point of view. Ontologies enable users to formally describe a domain of knowledge in terms of concepts and relations among them and hence they can be used for unambiguously specifying the information captured by the relational database. However, accessing information residing in a database using ontologies is feasible, provided that the users are keen on using semantic web technologies. For enabling users form different disciplines to retrieve the appropriate data, the design of a Graphical User Interface is necessary. In this work, we will present an interactive, ontology-based, semantically enable web tool that can be used for information retrieval purposes. The tool is totally based on the ontological representation of underlying database schema while it provides a user friendly environment through which the users can graphically form and execute their queries.

***Keywords*—**Ontologies, Relational Databases, SPARQL, Web Interface.

I. Introduction

THE relational database (DB) management systems provide a stable and robust solution for storing and managing a large amount of data while they can efficiently handle user queries. Access to the underlying data is most often provided to end users and non IT experts in general through customised forms and interfaces which are tailored to the needs of the application they serve. However, in the background the queries applied are SQL-based [1] and, hence, the searching mechanisms are based on characters (or even bytes) matching techniques. This, in turn, poses a series of restrictions to both developers and end users, especially in cases that complicated queries, which might be semantically expressed in a different, broader or narrower way than the data at the database(s) and are, thus, isolated from the structure and vocabularies of the underlying data source(s), need to be applied to the latter. Hence, either developers are forced into placing great programming effort in order to implement mechanisms which process the submitted queries in order for the latter to be (fully or partially) applicable to the underlying database or end users are forced to fully comply with the underlying vocabularies with a great risk in having a narrower view of the available data.

Ontologies, which can be formally represented through RDF Schema (RDFS) [2] and Web Ontology Language (OWL) [3] among others, enable users to formally describe the concepts of a domain as well as the relations among them. They provide a variety of constructors for expressing meaningful constraints [4] such as organising the concepts in hierarchies or specifying cardinality restrictions about properties defined. Based on these key features, ontologies can provide rich representations of the information available in a relational database and can be used instead of or in parallel with the latter, providing a *formal description* of the information it contains, by introducing additional *constraints* and/or relations that cannot be expressed in the relational model, while they can also support *knowledge inferencing*. However, accessing the data stored in databases using ontologies requires adequate knowledge of Semantic Web technologies and especially OWL and SPARQL [5]. Thus, for allowing users from different disciplines with limited IT background (e.g., clinical experts) to benefit from ontology-mediated access to relational databases, a Graphical User Interface (GUI) is essential.

Despite the fact that a large corpus of research work has been published so far on ontologies, including but not limited to ontology representation languages, ontology evolution and alignment, much fewer efforts have been reported on accessing their data, especially through a GUI. To our knowledge, there is no available tool that enables non-IT users to graphically form a SPARQL query based on the information provided in an OWL ontology or an ontology on top of a relational database. In this work, we present an interactive, semantically enabled, user friendly web tool which allows users to *graphically specify* the information they are looking for, including the condition(s) that the data should satisfy. In the background, the system *automatically generates* the appropriate SPARQL queries, which are then being used for information retrieval purposes.

More precisely the document is structured as follows. In Section II we provide related work regarding ontology-mediated access to relational databases. In Section III, we

Efthymios Chondrogiannis is with the National Technical University of Athens, 15773, Greece (phone: +30 210 7722132; fax: +30 210 7722132; e-mail: chondrog@mail.ntua.gr).

Vassiliki Andronikou, Efstathios Karanastasis, and Theodora Varvarigou are with the National Technical University of Athens, Zografou, 15773, Greece (e-mail: vandro@mail.ntua.gr, ekaranas@mail.ntua.gr, dora@telecom.ntua.gr).

briefly present the approach being followed for enabling users to graphically form and execute their queries. In Section IV the Web Application developed for information retrieval purposes is being presented in detail and Section V includes a description of the background mechanisms. The discussion about the tool follows in Section VI and, finally, in the last section the work presented is summarised.

II. Related Work

The potential use of ontologies in combination with relational database management systems has been extensively studied so far and a considerable number of systems and frameworks have been published, included but not limited to MASTRO [6], OptiqueVQS [7] and Quest [8]. The systems are being classified in two broad categories. In the first category belong systems which produce an ontology based on the information existing in a data source, which accordingly is being used for evaluating user queries. The second category encompasses systems such as D2R [9], which provide only an ontological representation of the relational database's schema while user data remain in the database. Each one of the two approaches has its own advantages and weaknesses. For example in the first one, user queries are directly executed, whereas in the second case query (and in some case results) translation is necessary. Furthermore, the first approach cannot properly handle cases involving a large amount of data, due to the excessive resources (e.g., memory) required.

Concerning the Ontology being used, it may has been either *extracted* from the DB schema or *designed* by the end users. Regardless of the process being followed for its design, *mapping* among the terms of the ontology and the DB schema is required. In the first case, the mappings are automatically generated by the system [9]. On the other hand, the ontological elements specified in the extracted ontology are explicitly based on the DB schema lacking thereof of a detailed semantic representation and requiring support from the DB experts in order to clarify the names and the values within the target DB. In the second case, the ontology designed provides a thorough and meaningful semantic description of the underlying concepts and data. However, the mappings should be manually specified by the end users, a process which requires rather great effort (given the need to cope with syntactic, structural and semantic heterogeneity) which increases exponentially with the size of the ontology.

For the automated ontological representation of database schema, in general, for each table a class with the corresponding Object or Datatype properties is generated based on their fields [10]. A different approach presented in the paper [11], according to which for each table all possible sub-classes (including object properties that point to subclasses) are also produced taking into account the values in the fields presented and possible subgroups they can form. However, in the second approach much more human effort is required for the design and development of the corresponding ontology taking into account the large number of entitles produced and the limited contribution of the relational database schema and values, from the conceptual point of view. This stems from the fact that often the names of tables and fields are not so meaningful (e.g., may provide an abbreviation of a term rather than its long form), while they may also lack of formal description.

Concerning the formulation of SPARQL queries, there are a few publicly available *editors* such as Flint [12], OpenLink Virtuoso [13] and Twinkle [14]. The editors facilitate users for *writing* a SPARQL query highlighting keywords and verifying the validity of SPARQL queries formed. However, they cannot be used by the users which are not familiar with Semantic Web technologies and non IT experts in general.

For the SPARQL Query Formulation there are only a limited number of publicly available *visual tools*. Pubby [15] is a Linked Data Frontend for SPARQL Endpoints. It enables users to explore data sources following the links presented. Nevertheless, it does not allow users to form queries specifying the conditions the data should satisfy based on their properties. The rdf:SynopsViz [16] is a framework for hierarchical charting and exploration of Linked Open Data. This tool focuses on the classification of terms rather than the internal structure of data. In fact the classification of terms is an important parameter when searching in ontologies. However, ontologies provide a lot of information in terms of properties and hence the path to be followed for retrieving the corresponding data should be precisely determined. In the SparqlFilterFlow [17] it is described the process being followed for the SPARQL query formulation. However both the web interface and the interactions with the end user are not adequately described.

For querying either ontologies or relational databases *ad-hoc* web interface has been also developed which they typically provide a set of forms with a few input fields each depending on the specific purposes they serve. For example, the GUI provided by PAT [18] enables users to retrieve the number of eligible patients based on the eligibility criteria specified. It should be noted that the process being followed for the design of such interfaces is rather manual and hence it requires a considerable amount of time and human resources.

III. Overall Approach and Methodology

The proposed solution for allowing users to apply complex queries - which are isolated from the structural and vocabulary-related details of the underlying data source - to a relational database is based on the ontological representation of the target database. Nevertheless, ontologies automatically generated from databases are solely based on the latters' schema, which is a formal description of the data structure, and, hence, they are not sufficient for our purpose. Consequently, the design of an ontology which further encapsulates the meaning of data is necessary. However, when dealing with large data sources which contain millions or billions of records, performance and administrative reasons impose maintaining the data in the relational database. Hence, accessing the data requires the transformation of SPARQL queries (as initially formulated based on the ontology on top of the database) to the corresponding SQL ones. The following paragraphs provide a step-by-step presentation of

the overall approach followed along with the technologies incorporated and/or developed (Fig. 1).

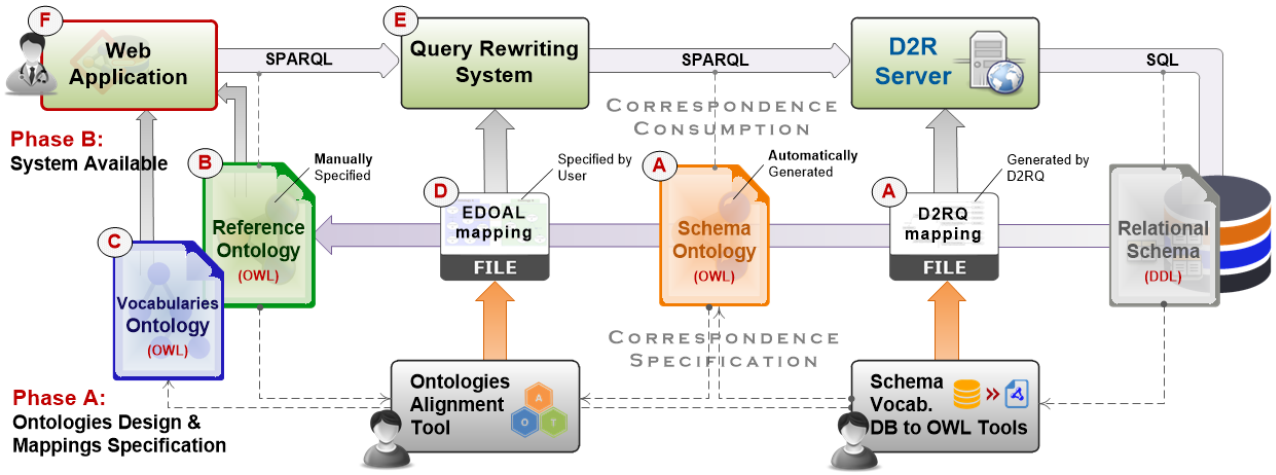

Fig. 1 The Steps we should follow for Linking the Web Application with a Relational Database

A. Automatically Generated DB Schema

Initially, an ontological representation of the database schema (Schema Ontology) is extracted, based on the DB 2 OWL algorithm [10]. In general, this algorithm automatically generates an OWL class for each table and an Object or Datatype property for each field based on their definition. More precisely, in case a field is a foreign key, an Object property is being produced, otherwise a Datatype Property is generated, the range of which depends on the corresponding SQL datatype. Also the mapping between the ontology and schema is automatically generated using the D2RQ platform [19]. It should be noted that, the automatically produced elements of Schema Ontology are explicitly based on the information specified in the database (i.e., name and description) and, hence, may lack their formal description. For this purpose, for each ontological element, especially when not being adequately described, the appropriate label and comments needs to be introduced, in close collaboration with the DB experts.

B. Design of Database Reference Ontology

In order to provide a real conceptualisation of the domain that the database covers using a limited number of elements that the end users can better understand and use for expressing their queries, the Reference Ontology has been developed, the design of which was driven by the Schema Ontology. More specifically, classes have been organised into three categories (see Fig. 2). The first one, named “Data”, encompasses the classes that are being used for capturing the parameters for the entities of their interest (e.g., Diseases, Drugs, in the case of a database which captures the Electronic Health Records of patients). The second one, named “Vocabularies”, includes the classes that are being used to refer to a controlled set of terms (e.g., ICD 10 Code in the case of a biomedical ontology which captures diseases among others). In the third category, named “Complex Data Types”, there are the classes which represent data that are being captured by one or more properties that should be examined together, such as Quantity and Period of time. Concerning Object and Datatype properties, they have been organised in broader categories, based on the information they carry whereas properties with the same meaning have been replaced by one, including but not limited to those properties that uniquely determine an element (id), codes (symbols) and their name (human readable description).

C. Design of Vocabularies Ontology

In order to allow information retrieval which does not require the end users to know the codings and vocabularies which are used in the database, the controlled set of terms used is extracted in another ontology; the “Vocabularies Ontology”. This ontology contains both the international classification systems being used (downloaded from the web, since their formal description is often missing from the database) and any locally defined controlled set of terms. The latter are often stored in separate tables in which the rest of database entities refer to. However, in some cases, the values presented in the database tables can also serve as vocabularies with specific meanings, depending on the table and/or field in which they appear. For instance, if the database stores patient data from a healthcare entity which include information about the person’s current participation in any clinical trial, then the value ”X” in the field “CT” may indicate that the person already participates in another clinical trial (CT), while the value null may indicate the opposite. In such cases, further interpretation of the appearing values (including null) is necessary. For extracting the aforementioned database (DB) vocabularies in an OWL ontology, the DB-to-OWL tool [20] has been developed. The ontology produced can be further processed by domain experts (e.g., in order to organise terms in categories) using Protégé [21].

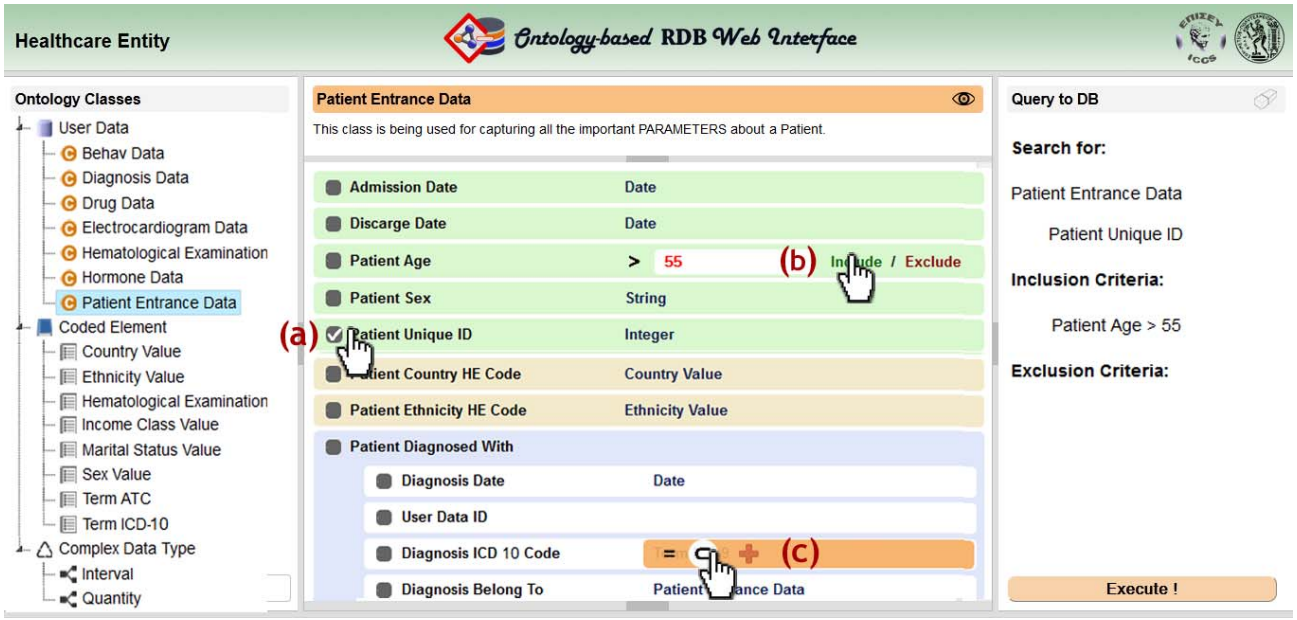

Fig. 2 A screenshot from the Ontology-based Web Interface for patient data as captured by a Healthcare Entity. (a) Selection of the parameters we should present for eligible entities, (b) Specifying an age criterion - in advance we have selected that it should be greater than the given value, (c) Pop-up window presented for introducing a diagnosis code restriction

D.Mapping Reference Ontology with Schema Ontology

For accessing information residing in a relational database through the use of Semantic Web technologies, mapping among the terms of the Reference and Schema ontologies is necessary. In general, there is a variety of mismatches among the terms of semantically overlapping ontologies [22]. However, given the specific development process described above, when mapping the Reference Ontology with the Schema Ontology, only specific types of mismatches occur. More precisely, since a few properties with the same meaning have been replaced by a single one, when mapping the corresponding properties the domain in which the entities belong to should be taken into account. Also, in cases in which two or more properties (e.g., value and unit) have been replaced by a single one (e.g., quantity) the correspondence among the group of them should be determined. Finally, for covering cases in which a datatype property has been replaced by an object property that points to a controlled set of terms, the path formed should be mapped with the corresponding datatype property. For specifying the correspondence among the aforementioned entities, we have developed and used the Ontologies Alignment Tool [23], while the mappings rules specified have been exported in an XML file, based on the Expressive and Declarative Ontology Alignment Language (EDOAL) [24].

E. Incorporation of Semantic Web Technologies

The aforementioned mapping is used at run-time (i.e., when a query is submitted) in order for the applied queries expressed based on the Reference Ontology to be rewritten based on the Schema Ontology. The rewritten SPARQL queries are, in turn, translated into the corresponding SQL queries by the D2R server and used for retrieving the appropriate data from the target relational database. It should be noted that, the query rewriting/translation process performed is based on the EDOAL (defined by user) and D2RQ (automatically generated) mapping files, respectively. In brief, during the SPARQL to SPARQL rewriting process, for each correspondence specified the system automatically produces the appropriate transformation rule which determines the changes that should be applied to the SPARQL query. Accordingly, the system detects the mapping rules that should be enforced, based on the specific ontological elements within the SPARQL query provided. The mapping rules that can be fired are, then, placed in the correct order and are executed. The outcome of this process is a *semantically equivalent* SPARQL query expressed based on the terms of the Schema Ontology. A detailed description of the query rewriting mechanisms has been presented in our previous work [25].

F. The Graphical User Interface

The provision of a user friendly and easy to use means for accessing a database in an ontology-mediated way, as described so far, introduces the need for a GUI, which in this case constitutes an innovative Ontology-based Query Generation Web Application. This tool allows users to graphically specify the data of their interest and the conditions that they should satisfy. In the background, a series of mechanisms has been implemented through which the user can determine the aforementioned parameters based on the Reference and Vocabularies ontologies and a completely automated SPARQL query is produced for information retrieval purposes. The generated SPARQL query is, then, executed using the Query Rewriting System and the data

retrieved from the relational database are presented through the web tool.

IV. The Web Application

A. Functionality

The Ontology-based Query Generation Web Application (OntoQGenApp) provides an interactive environment by means of which the end users are able to easily navigate through the elements specified in the Reference Ontology and graphically formulate their queries. More precisely, the end users are able to define the *type* (i.e., class) of data they are interested in, their *characteristics* (i.e., properties) to be evaluated and the *condition* (i.e., range or set of "acceptable" values) they should satisfy.

Fig. 2 presents a screenshot of the OntoQGenApp through an example of accessing anonymised patient data stored in the relational database of a Healthcare Entity. The classes defined in the Reference Ontology are presented in the left side of the screen and the properties that can be applied to an instance of such a class are presented in the middle of screen. Using the elements of this panel, the user can graphically formulate their queries, which are then summarised in the right side of the screen (and internally being stored in a JSON message – see Fig. 5). For user convenience, the conditions that the data should satisfy have been separated into two broad categories named "inclusion" and "exclusion" (I/E) criteria; i.e., the data returned as a response to user queries should satisfy all the inclusion criteria and, in parallel, they should not satisfy any of the conditions specified in the exclusion ones.

B. Web Tool Architecture

Fig. 3 depicts the main components of the OntoQGenApp developed. It consists of a web interface through which the end users can graphically express their queries. In the background, the tool utilises the services provided by Requests Handler component.

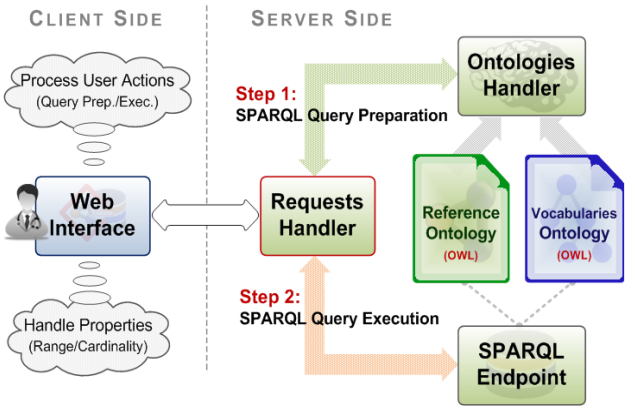

Fig. 3 The Web Tool (OntoQGenApp) Architecture

TABLE I
Functionality provided by Requests Handler component

ID	Name	Input	Output	Description
F1	Classes		Classes (Tree)	Provides a JSON message with the OWL classes (placed in such a way that they form a tree) specified in the Reference Ontology.
F2	Properties	Class (URI)	Properties (List)	Provides a JSON message with the properties (name, description, range, axioms) from the Reference Ontology that can be applied to an instance of the OWL class specified.
F3	Suggested Terms	String	Terms (List)	Provides a JSON message with the terms specified in the Vocabularies Ontology based on the sequence of characters (string) provided.
F4	Execute Query	Query Data	Results	Prepares and Executes the appropriate SPARQL query based on query JSON data provided. After the execution of the query, the data are presented.

Table I summarises the functionality provided by the Requests Handler according to the user's actions. In the first three cases (Table I, F1-F3) the Requests Handler uses the Ontologies Handler component for retrieving the appropriate information from the Reference and Vocabularies ontologies (i.e., Classes, Properties and Suggested Terms). In the last case (Table I, F4), the Ontologies Handler component initially produces the appropriate SPARQL query containing the data specified by the end user, and executes it for retrieving the relevant data from the relational database.

In the process that generates the SPARQL query; special attention should be given in the conditions in which a semantic operator is being used. A *semantic operator* is an operator for the evaluation of the boolean expressions formed we should take into account the *meaning* beyond the sequence of characters provided. For instance, the operator "⊃" indicates that any term with narrower meaning than the one specified is valid. Taking into account that the evaluation of SPARQL queries is based only on the triples that have been directly asserted in the RDF graph [26], in order to retrieve all the semantically correct results without changing the semantics of SPARQL, the tool initially retrieves all terms with broader, narrower or the same meaning based on the semantic operator specified and then includes them in the SPARQL query executed.

The Ontologies Handler component has a distinctive role in the developed tool. It is responsible for communicating with the Reference and Vocabularies ontologies (loaded in the tool) and provide the data requested by the Requests Handler. The OWL classes provided (Table I, F1) are organised in a hierarchy based on the axioms specified. Concerning the properties "available" for each class (Table I, F2), apart from their label (or local name, if the label is not available) and URI, the component also provides their Range and Cardinality Restrictions (either explicitly specified or inferred). In the case of coded elements (i.e., terms specified in the Vocabularies Ontology), for retrieving semantically relevant terms (in this case, with either similar or narrower meaning) it initially

creates the Inferred Model based on sub-class axioms specified using the Pellet reasoner [27], and, then, retrieves the appropriate data using SPARQL queries. For instance, the following SPARQL query is executed for retrieving all ICD-10 [28] terms (i.e., code and label) with a narrower meaning than the given one (ICD-10 code):

```
PREFIX rdf:  <http://.../rdf-syntax-ns#>
PREFIX rdfs: <http://.../rdf-schema#>
PREFIX icd10cm: <http://.../ICD-10-CM.owl#>

SELECT ?subcode ?sublabel
WHERE {
      ?cls rdf:type icd10cm:ICD-10-CODE .
      ?cls icd10cm:code {icd10-code}    .
      ?subcls rdfs:subClassOf ?cls      .
      ?subcls icd10cm:code ?subcode     .
      ?subcls rdfs:label ?sublabel
}
```

In this SPARQL query, the entity placed within curly brackets should be replaced by the specific ICD-10 code each time.

C. Interaction among Components

Fig. 4 presents the interaction between the client and server side components. As presented, the formulation of user queries takes place on the client side, whereas the production of the SPARQL query lies on the server side based on data recorded. The graphical formulation of the queries is based on the interactive environment of the tool (Fig. 2) through which the users are able to specify the properties of the data (using the button on the left side of each one – Fig. 2 (a)) as well as the conditions that the data should satisfy, following the options appearing on their screen next to each property (Figs. 2 (c) and (b)). The latter are explicitly based on the definition of the corresponding ontological elements in the Reference Ontology as well as the controlled terminologies provided. A detailed description of the background mechanisms for query formulation and the algorithm for generating the corresponding SPARQL queries is provided in section follows.

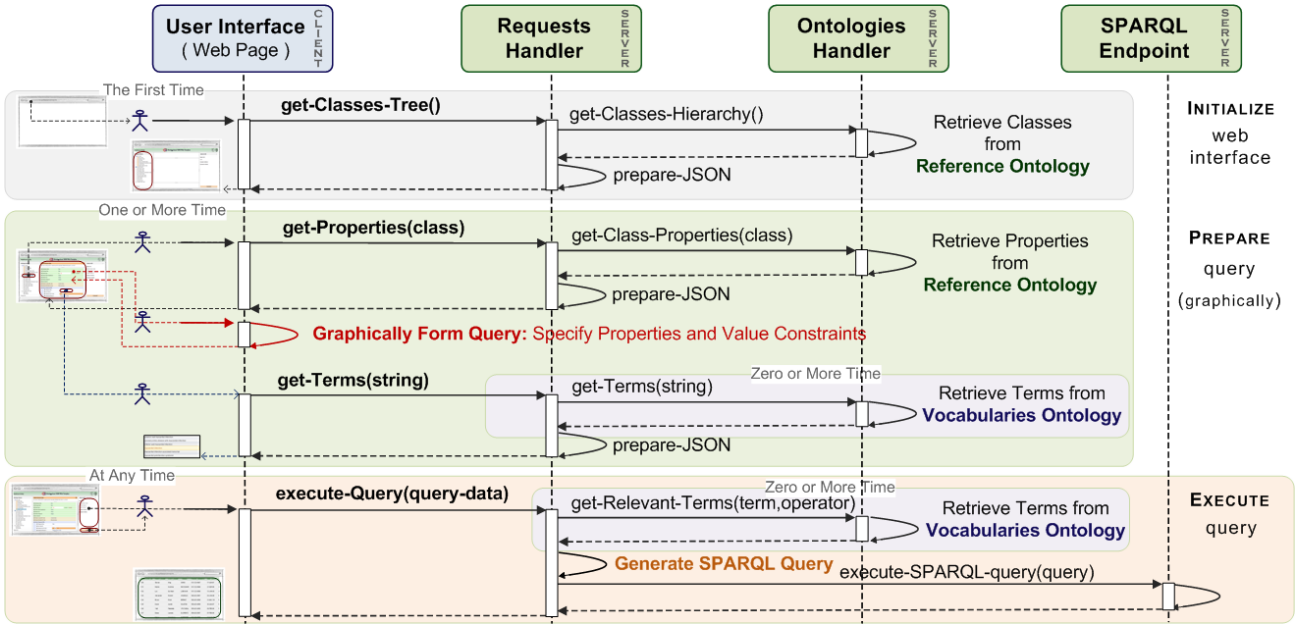

Fig. 4 The interaction among client and server side components

V. Background and Mechanisms

A. Model-Based Query Formulation

The Reference Ontology, as already mentioned, provides a formal description of the parameters captured for each entity. Consequently, by using the Reference Ontology we can explicitly determine both the data we are looking for (specific properties) and the conditions that their values should satisfy. In fact, both of them are tightly linked with the type of the properties' value (i.e., Range) as well as the cardinality restriction (or generally axioms) specified.

The *range* of a property determines whether a property-value "restriction" or property-value "expansion" (i.e., retrieval of properties based on its range) process will be held. In case their value is being captured by a *primitive data type* (e.g., integer) the range of values in which the values of our entities should belong to (property value restriction) can be specified. The available restrictions that can be applied depend on the specific datatype of each property. For example, when the value of a property is being captured through a number, a comparison operator can be used, including "greater-than" and "equals_to", while the conditions formed can be combined with a logical operator ("and", "or", "not") for specifying more complex restrictions. For the sake of simplicity, users are also allowed to determine whether the value is within a

specific set or range (especially for integers), despite the fact that the aforementioned restrictions can be also specified from the proper combination of *logical and comparison operators*.

When the value of a property is being captured by a *complex data structure*, the available properties should be examined first and, then, desired restriction(s) as well as their logical relation be specified. For example, for specifying a criterion based on diagnosis data, the parameters (such as disorder and date clinically identified) for each entity should be first examined and, then, the desired values are determined. It should be noted that in case more than one "range classes" are available (e.g., any subclass from the one specified), the users should first select the appropriate "range class" and, then, specify the criteria based on the properties of the class selected.

In case the range of a property is of a "compound" data-type such as code elements, quantities and period of time, a similar approach can be followed. More precisely, the available properties are examined and, then, the appropriate value restrictions are set. However, taking into account the meaning of properties which often appear together in queries as well as users' needs for a simple interface, an alternative approach is being also provided. More precisely, for each one of them the available operators as well as the parameters that should be provided in each case have been determined. For instance, in the case of a "coded element", the users can specify that they are interested in entities with exactly the same sequence of characters as the given ones (1 parameter required) or search for entities with the same, broader or narrower meaning (using semantic operators). In the case of "quantities" the users can specify a boolean expression using the comparison operators. Nevertheless, in this case, they should determine 2 parameters; both value and unit(s) of measurement.

The *cardinality restriction* (CardinR) of properties is another important factor when specifying the conditions the data should satisfy. The CardinR may have been explicitly specified in the definition of an OWL class or implied based on the definition of the properties.

In our work, we would like to highlight three different kinds of CardinRs. The CardinR "max 1" indicates that an entity either has a property or not; hence is *optional*. For this purpose, we have introduced an additional operator named "has-value" for detecting those entities for which there is a specific value in the corresponding property, but it does not satisfy any further restriction. The CardinR "min 1" indicates that an entity has a least one instance of such a property. Consequently, when specifying a restriction, we may possibly define that we are looking for entities which satisfy "any" of the values presented or "all" of them. For instance, retrieving those patients diagnosed with both cancer and arthritis. In general, when more than one instances of a property may appear, more than one *conditions* (i.e., range or set of values) can be provided in the restrictions formed, which are being semantically linked with an "and" operator.

Special attention should be given in the CardinR "min 0" (properties by default have this CardinR, unless any other axiom is specified). Absence of this property may denote either that the individual does not have any instance of this property or that there are one or more instances but they are not provided. In order to better understand this kind of relation, one should consider the scenario in which a person is associated with zero or more diagnoses. In case a specific person is not associated with any diagnoses, the corresponding person may either have never had any health problem or the person's diagnoses have not been recorded in the database (missing data). Taking into account the semantics of the SPARQL language which is based on the close-world assumption [29] (relational databases also follow this approach); an entity not having any of the aforementioned properties simply denotes that the entity does not have such data. Consequently, in the evaluation of the formed queries, negation as failure [30] will be used.

B. Automatic SPARQL Query production

The SPARQL query to be applied for information retrieval purposes is automatically generated by the system based on the data provided in JSON format (Fig. 5).

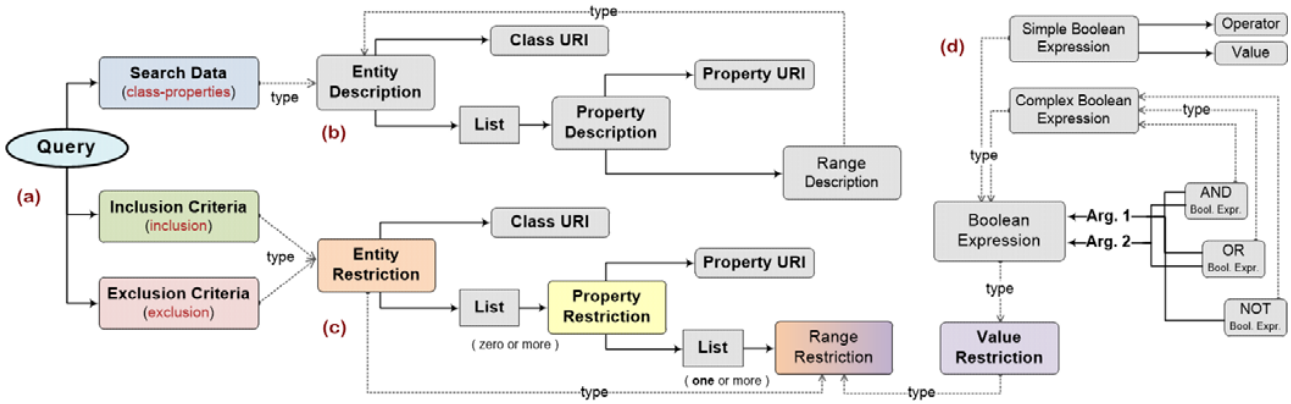

Fig. 5 (a) Parameters recorded for each user-defined query, (b) Data presented for the eligible entities, (c) The restrictions with which the entities should comply, (d) The conditions that the values of properties should satisfy

The production part of a SPARQL query (i.e., SELECT clause) determines the information sought after; hence it contains the variables corresponding to properties selected. The restrictions part of the query (i.e., WHERE clause) determines the "path" that should be followed for detecting the parameters of interest as well as the conditions that should be satisfied.

In the WHERE clause of the generated query, the triple patterns should be included which specify the meaning of the variables presented in the SELECT clause (GraphPattern). Also, taking into account the fact that conditions are categorised into positive and negative ones (i.e., inclusion and exclusion criteria), the WHERE clause includes two additional main blocks; one for positive conditions (which should be satisfied) and another one for negative conditions (which should be not satisfied). In both cases they consist of one or more triple pattern(s) and FILTER clauses (GroupGraphPattern) that correspond to the restrictions provided. However, in the case of negative conditions the entities should not satisfy not even one of the respective triple patterns and FILTER clauses. This is being specified in the generate SPARQL query by using the "FILTER NOT EXISTS" operator, available at the latest version of SPARQL (version 1.1 [31]). Alternatively, the "OPTIONAL FILTER !BOUND" pattern can be used, in which case the variables present in the "not-bound" clause should be additionally determined.

In the following lines, the overall structure of the generated query is provided. It should be noted that the three graph patterns present in the WHERE clause of the SPARQL query share the same variable, so that the conditions specified are applied to the same entities.

```
SELECT List-of-Variables-for-Parameters-Selected
WHERE {
      GraphPattern (variable, search-data)    .
      GroupGraphPattern (variable, inclusion-criteria)
      FILTER NOT EXISTS {
         GroupGraphPattern (variable, exclusion-criteria)
      }
}
```

The most interesting part in the SPARQL query production is the process that specifies the restrictions that an entity (i.e., variable) should satisfy. This process receives as input one or more conditions and returns one or more triple patterns accompanied by one or more FILTER clauses (i.e., a GroupGraphPattern). In Fig. 6, the algorithm being used in pseudo-code is presented. The algorithm distinguishes between two types of conditions: a. Entity Restriction and b. Value Restriction. The Entity Restriction specifies the type of entities along with the properties the value of which has been restricted to a specific set or range (e.g., being a Patient the age of which is within the range provided, while they have been also diagnosed with the data that satisfy the conditions follow). The Value Restriction specifies the conditions that the values of properties should satisfy (e.g., being older than 55 and diagnosed with Myocardial Infarction). It should be noted that a Datatype Properties (e.g., age of a person) is being followed by a Value Restriction whereas an Object Property (e.g., diagnoses data of a person) by an Entity Restriction.

```
1  function  create-Group-Graph-Pattern ( variable , restriction ) {
2      IF ( restriction IS-A Value-Restriction ) {
3          IF ( restriction IS-A Simple-Boolean-Restriction AND restriction.operator == "has-value")
4              return ""; // returns nothing !
5          ELSE // return a FILTER clause with a boolean expression based on operators and values used
6              return FILTER( condition-For-Variable(variable, restriction) );
7      } ELSE { // IF ( restriction IS-A Entity-Restriction )
8          Class-URI = restriction.class-URI;
9          Properties-Restriction-List = restriction.List;
10         // Determine Entity Type
11         GraphFilterClause = variable rdf:type Class-URI ;
12         // Determine Entity Properties Conditions
13         For each ( Property-Restriction IN Properties-Restriction-List ) {
14             Property-URI = Property-Restriction.property;
15             Range-Restriction-List = Property-Restriction.List;
16             // In case more than one conditions provided
17             For each ( Value-Restriction IN Value-Restriction-List ) {
18                 GraphFilterClause += variable Property-URI new-variable ;
19                 GraphFilterClause += create-Graph-Filter-Clause(new-variable, Value-Restriction);
20             }
21         }
22         return GraphFilterClause;
23     }
24 }
```

Fig. 6 The algorithm for Generating the appropriate Triple Patterns and Filter Clauses based on conditions specified

The generated GroupGraphPatterns depend on the type of restrictions specified for each one of I/E criterion. In case of an Entity Restriction, the system specifies the class in which entities should belong to (Fig. 6, Line 11) and accordingly places the appropriate triple pattern based on the defined properties. At this point, it should be noted that in case more than one conditions are provided (e.g., diagnosed with both cancer and arthritis), the system introduces a different triple

pattern for each one (Fig. 6, Line 18). Concerning Value Restrictions, the system introduces the appropriate FILTER clause based on the boolean and comparison operators specified. Special attention should be given when a "has-value" or semantic operator is being used. In the first case, no FILTER clause is being introduced (Fig. 6, Line 3), whereas, in the second one, the system initially finds the appropriate terms based on the semantic operators specified and includes them in the FILTER clause (functionality provided by conditions-For-Variable service – Fig. 6, Line 6).

In the WHERE clause of the SPARQL query, apart from the triple patterns which correspond to I/E criteria, we should also include the triple patterns that regard the properties that are requested to be retrieved, as mentioned before. However, in this case, the variables introduced are not part of any FILTER clause. Also one triple pattern for each property is specified, even if more than one instance occurs. Moreover, in case a property is either optional (CardinR is max 1) or its cardinality restriction may be zero (e.g., CardinR is min 0), we can also use an OPTIONAL clause so that we do not exclude entities which do not have the corresponding property.

Concerning the generated SPARQL query, we can reduce its complexity by removing unnecessary triple patterns, taking into account the ones specified for the I/E criteria as well as their cardinality restriction(s). For example, in case we are looking for an age property (with cardinality restriction maximum 1) for which we have already defined a restriction (hence, the corresponding triple pattern has already been provided), there is no need to introduce a new triple pattern in the WHERE clause. However, if we are looking for the diseases from which the patients suffer (with cardinality restriction being zero or more) for which we have already defined a restriction, an additional triple pattern(s) that correspond(s) to patients conditions should be introduced in the WHERE clause. This is necessary since if we use the variables presented in the triple patterns generated for inclusion/exclusion criteria, we will retrieve only those problems satisfying restrictions provided and not all the available ones.

VI. Discussion

When there is the need to specify a variety of conditions, the graphical user interface will be inevitably rather complicated. However, by following a model-based approach for criteria expression, we are focusing on the meaning of available information and the restrictions that should be satisfied. In contrast with other approaches which may provide a specific set of forms for the user to fill in, such as the one presented in PAT for eligibility criteria specification, in this work we dynamically form only the restrictions that are necessary for the query formulation, keeping, thus, the web interface as simple as possible. More specifically, the proposed tool does not present in advance a form for the provision of the target values for a property (e.g., age) but rather presents the property and its type (e.g. integer) and the users only interact with the corresponding input fields if they decide that they wish to define a restriction regarding this property. Especially when the value of the property captured is represented by a complex data structure (e.g., diagnoses data), users can additionally utilize the elements presented by the tool for specifying the set or range of appropriate values for the variable. When the value of a property is captured by a coded element, quantity or period of time, the web interface allows users to easily handle the property as if it were a primitive data type without increasing the complexity of the GUI.

The GUI presented allows users to specify in detail the target data; a functionality which is not provided by other tools, such as the web interface provided by D2R server. Concerning other ontology-based web interfaces, such as rdf:SynopsViz, the GUI designed is much more advanced since it enables users to precisely determine the information they are looking for based not only on their classification but also their properties. On the other hand, the current version of the Web Tool produces only a subset of the possible SPARQL queries we can form. For example in this work we have assumed that all conditions specified for the properties of compound data types should be satisfied or not (e.g., inclusion criterion: diagnosed with Myocardial Infarction "and" the date diagnoses was in the previous 2 weeks).

In the tool presented, it should be noted that all queries specified are being translated to semantically equivalent SQL queries. This is feasible since the design of Reference Ontology was driven by the elements automatically generated in the Reference Ontology. Hence, all the ontological elements specified in the Reference Ontology are properly mapped (either 1:1 or n:m alignments) with the corresponding elements from Schema Ontology and accordingly database schema. This is very important, since when the database ontology is being arbitrary designed there might be mappings which are incomplete or partial due to the semantic distance between the ontology and the database. This, in turn, leads to cases that the users queries cannot be translated to the semantically equivalent DB queries, since some conditions have been ignored due to inability express them using the terms of DB schema.

Concerning the controlled vocabularies being used in the formulation of user queries used, they come from the Vocabularies Ontology, the design of which was driven by the controlled set of terms specified in the relational database. Alternatively, a new version of international classification systems (e.g., ICD 11) or widely accepted codings (e.g., HL7 Sex Codes [32]) can be used. Consequently, mapping among their terms (i.e., the ones used in the GUI and the ones specified in the database) is necessary, especially for translating user queries to the corresponding DB queries. The data retrieved, can be further processes translating the terms retrieved from the database to the corresponding ones, based on mappings specified. However, this process is optional, taking into account that data retrieved (e.g., name of problems diagnosed) will be further examined by domain experts.

VII. Conclusion

In this work we have presented an interactive web tool that

allows ontology-mediated access to relational databased in a user friendly manner. In the background the tool utilises the information in the Reference Ontology which constitutes both a structural and semantic representation of the target relational database and based on which the end users express their queries through the interactive, user friendly environment presented. During this process the OntoQGenApp facilitates the end users, by suggesting candidate terms based on the data specified in the Vocabularies Ontology while it also enables users to specify meaningful constraints using semantic operators. When all conditions have been specified, the system automatically generates the corresponding SPARQL query which is then translated into a SPARQL query expressed over the Schema Ontology, which has been directly derived from the relational database, and, in turn, produces the semantically equivalent SQL query to be applied to the target database.

Acknowledgment

This work is being supported by the OpenScienceLink project [33] and has been partially funded by the European Commission's CIP-PSP under contract number 325101. This paper expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this paper.

References

[1] D. D. Chamberlin, R. F. Boyce, SEQUEL: A structured English query language, In Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control (SIGFIDET '74), New York, USA, 1974, pp. 249-264, DOI: 10.1145/800296.811515.

[2] D. Brickley, R. V. Guha, RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (accessed Nov. 2014)

[3] D. L. McGuinness, F. V. Harmelen, OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/ (accessed Nov. 2014)

[4] P. Spyns, R. Meersman, M. Jarrar, Data modelling versus ontology engineering, in SIGMOD Rec., vol. 31, num. 4, December 2002, pp. 12-17, DOI: 10.1145/637411.637413.

[5] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF, W3C Recommendation 15 January 2008, http://www.w3.org/TR/rdf-sparql-query/ (accessed Nov. 2014)

[6] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, et al., The MASTRO system for ontology-based data access, Semant. web, vol. 2, num. 1 , January 2011), pp. 43-53.

[7] A. Soylu, M. Giese, E. Jimenez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, OptiqueVQS: towards an ontology-based visual query system for big data, In Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems (MEDES '13). ACM, New York, NY, USA, 2013, pp. 119-126, DOI: 10.1145/2536146.2536149

[8] M. Rodríguez-Muro, D. Calvanese, Quest, a System for Ontology Based Data Access, In OWLED 2012, 2012.

[9] D2R Server: Accessing databases with SPARQL and as Linked Data, available at http://d2rq.org/d2r-server (accessed Nov. 2014)

[10] N. Cullot, R. Ghawi, and K. Yétongnon, DB2OWL: A Tool for Automatic Database-to-Ontology Mapping, In Proceedings of 15th Italian Symposium on Advanced Database Systems (SEBD 2007), 2007, pp. 491–494.

[11] K. Munir, M. Odeh, P. Bloodsworth and R. McClatchey, "Using Assertion Capabilities of an OWL-Based Ontology for Query Formulation", 3rd International Conference on Information & Communication Technologies: from Theory to Applications (ICTTA), IEEE, Damascus, Syria, 2008.

[12] Flint SPARQL Editor, available at http://cliopatria.swi-prolog.org/flint/index.html (accessed Nov. 2014)

[13] OpenLink Virtuoso SPARQL Query Editor, available at http://demo.openlinksw.com/sparql/ (accessed Nov. 2014)

[14] Twinkle: A SPARQL Query Tool, available at http://www.ldodds.com/projects/twinkle/ (accessed Nov. 2014)

[15] R. Cyganiak, C. Bizer, Pubby - A Linked Data Frontend for SPARQL Endpoints, available at http://wifo5-03.informatik.uni-mannheim.de/pubby/ (accessed Nov. 2014)

[16] N. Bikakis, M. Skourla, C. Papastefanatos, "rdf:SynopsViz - A Framework for Hierarchical Linked Data Visual Exploration and Analysis", available at http://83.212.97.83:8084/ (accessed Nov. 2014)

[17] F. Haag, S. Lohmann, T. Ertl, SparqlFilterFlow: SPARQL Query Composition for Everyone, The Semantic Web: ESWC 2014 Satellite Events, 2014, pp. 362-367, DOI: 10.1007/978-3-319-11955-7_49.

[18] A. Tagaris, V. Andronikou, E. Karanastasis, E. Chondrogiannis, C. Tsirmpas, T. Varvarigou, D. Koutsouris, PAT: an intelligent authoring tool for facilitating clinical trial design. Stud Health Technol Inform., 2014, pp. 205:970-4.

[19] C. Bizer, and A. Seaborne, D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs, in 'ISWC2004 (posters)', 2004

[20] DB to OWL Tools, available at http://ponte.grid.ece.ntua.gr:8080/DbToOwl/ (accessed Nov. 2014)

[21] Protégé, available at http://protege.stanford.edu/ (accessed Nov. 2014)

[22] M. Klein, Combining and relating ontologies: an analysis of problems and solutions, In IJCAI-2001 Workshop on Ontologies and Information Sharing, Seattle, WA, 2001, pp. 53-62.

[23] E. Chondrogiannis, V. Andronikou, E. Karanastasis, and T. Varvarigou, An Intelligent Ontology Alignment Tool Dealing with Complicated Mismatches, Accepted for SWAT4LS Workshop 2014.

[24] EDOAL: Expressive and Declarative Ontology Alignment Language, available at http://alignapi.gforge.inria.fr/edoal.html (accessed Nov. 2014)

[25] E. Chondrogiannis, V. Andronikou, K. Mourtzoukos, A. Tagaris, and T. Varvarigou, A novel query rewriting mechanism for semantically interlinking clinical research with electronic health records, In Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics (WIMS '12), ACM, New York, USA, 2012, pp. 48:1-48:12, DOI: 10.1145/2254129.2254189.

[26] M. Arenas, J. Perez, Querying semantic web data with SPARQL, In Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '11), ACM, New York, NY, USA, 2011, pp. 305-316, DOI: 10.1145/1989284.1989312.

[27] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, Pellet: A practical OWL-DL reasoner. in Web Semantics: Science, Services and Agents on the World Wide Web, vol. 5, num. 2, 2007, pp. 51-53, DOI: 10.1016/j.websem.2007.03.004.

[28] International Classification of Diseases (ICD), 10th version, available at http://apps.who.int/classifications/icd10 (accessed Nov. 2014)

[29] G. Bossu, P. Siegel, Saturation, nonmonotonic reasoning and the closed-world assumption, Artificial Intelligence, vol. 25 num. 1, Jan. 1985, pp. 13-63, Jan. 1985, DOI: 10.1016/0004-3702(85)90040-2.

[30] K. L. Clark, Negation as Failure, Logic and Data Bases, 1978, pp. 293-322, DOI: 10.1007/978-1-4684-3384-5_11

[31] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013 (accessed Nov. 2014)

[32] Administrative HL7 sex Code System, available at https://phinvads.cdc.gov/vads/ViewValueSet.action?oid=2.16.840.1.114222.4.11.927 (accessed Nov. 2014)

[33] E. Karanastasis, V. Andronikou, E. Chondrogiannis, G. Tsatsaronis, D. Eisinger, A. Petrova, The OpenScienceLink architecture for novel services exploiting open access data in the biomedical domain. In: Proceedings of the 18th Panhellenic Conference on Informatics (PCI '14), ACM, New York, NY, USA, 2014, pp. 28:1-28:6.