An Extension of DIG 2.0 for Handling Bulk Data

Diego Calvanese and Mariano Rodríguez

Faculty of Computer Science
Free University of Bozen-Bolzano
Piazza Domenicani, 3
I-39100 Bolzano, Italy
{calvanese,rodriguez}@inf.unibz.it

Abstract. The research community has noted the need to retrieve the instance level of an ontology from bulk data stored in external data sources (e.g., a relational database), in order to delegate to the external source all aspects of the actual management of the data. To achieve this, several methodologies have been recently developed to represent and reason about what we call here Ontologies with Linking Axioms. However, existing DL reasoners cannot properly deal with such ontologies. Indeed, including the instance level in the communication with a DL reasoner can cause a heavy communication overhead, and goes against the requirement of delegating data management to the external source. To overcome these problems, we present here an extension to the DIG 2.0 Interface that allows for the specification and management of Ontologies with Linking Axioms. The extension is a general framework which can accommodate any type of data source and linking axiom through specific implementations. We also present an implementation aiming at representing axioms linking data sources managed by an RDBMS to an ontology.

1 Introduction

In several areas such as Data Integration [1], the Semantic Web [2], and Ontology-Based Data Access [3], the intensional level of an application domain is represented by an ontology that provides a conceptual view of the data maintained by the system, and through which clients access the system services. In a setting where the amounts of data involved are large, such as the ones mentioned above, it may be more convenient, or even necessary due to architectural constraints, to have the data itself managed by one or more external systems (e.g., a Database Management System), and link the ontology to such systems by means of suitable *mappings* [3, 4].

Indeed, this way of proceeding offers several advantages with respect to the one where both the intensional (i.e., the TBox) and the extensional (i.e., the ABox) levels of an ontology are under the direct control of the ontology management system (OMS) and the associated reasoner[1]: (*i*) when data management can be fully delegated to the external system, then one can rely on such a system to improve both *efficiency* in processing queries and *scalability* with the data; (*ii*) *physical independence*, since changing

[1] Note that current Description Logic reasoners might be considered as the simplest form of OMSs with associated reasoner.

the structure and organization of the underlying data does not require to change the ontology itself, but only the mappings between the ontology and the data; (*iii*) *flexibility and robustness*, since temporary unavailability of a source does not invalidate the whole query answering process; moreover, querying can be immediately resumed as soon as the data source becomes available again.

We observe that whether it is possible to exploit the above advantages and delegate query processing to an external system, while preserving soundness and completeness of the overall query answering process, strongly depends on the expressiveness of the language used to express the ontology [5].

Moreover, if the issue of client-system communication efficiency is considered, asking the clients to retrieve and transfer data from existing data repositories to the OMS might create a large communication overhead in which the client is involved. One possibility to avoid this is to delegate data retrieval to the OMS by telling it how and from where to retrieve the data.

The communication mechanism which is shaping the way in which users and external systems interact with OMSs, the *DIG Interface* [6], currently does not foresee these kinds of tasks even though there are systems already implementing these ideas (e.g., KAON2[2], QuOnto[3]).

In this article, we provide the definition of a standard for this kind of client-system interaction for OMSs, in terms of an extension to the DIG 2.0 interface. The extension allows for the specification of *linking axioms*, where each such axiom generically connects elements of an ontology to elements of an external data source. Following [4], the elements to connect are specified in the linking axioms by means of two queries, over the data source and over the ontology, respectively. The extension does not take any commitments with respect to the actual language(s) used in a linking axiom. The linking axioms definitions are kept abstract, and are instantiated with specific types of queries when needed. As an example and proposal, we also provide an instantiation of the extension that allows to map a generic DL ontology to a relational database.

The rest of the paper is structured as follows. In Section 2 we present the DIG Interface, in Section 3 we briefly discuss linking axioms in DLs. We then present the proposed extension to DIG 2.0, ad show an instantiation for the case of relational data sources.

2 The DIG Interface

The DIG interface is an effort carried out by the DL Implementation Group (DIG) to standardize interaction with DL reasoners. The specification defines the communication mechanism to which DL reasoners (i.e., DIG servers) and clients comply. The interface has been widely accepted and most well known ontology design tools and DL reasoners have already incorporated it.

The DIG 1.0 interface [6] was given as a set of XSD schemas that define messages that server and client exchange and the assumptions to which the DIG server

[2] http://kaon2.semanticweb.org/

[3] http://www.dis.uniroma1.it/ quonto/

should comply. The messages, divided in *Tells* and *Asks*, allowed the client to: query for the server's reasoning capabilities, transfer an ontology to the server, and ask standard DL queries about the given ontology, e.g., concept subsumption, satisfiability, equivalence, etc. The concept language used to describe DIG ontologies was based on the $\mathcal{SHOIQD}_n^-$ DL.

The current version, DIG 2.0 [7], clearly separates the internal model, presented in terms of UML diagrams, from the communication protocol. This allows for the implementation of the interface in communication protocols different from the original HTTP/XML based protocol, e.g., APIs in RPC, WSDL, etc. An XML based communication mechanism is still defined and can be obtained by directly deriving XSD schemes from the interface's UML model. The set of messages, called *Requests* and *Responses*, has been enriched, allowing for a more robust handling of ontologies. The concept language used in DIG 2.0 is OWL 1.1, implying full OWL compatibility. Additionally, a clear mechanism for defining *extensions* to accommodate extra functionality in DIG servers is also provided.

The DIG 2.0 Interface is divided in the *core* specification and a set of *official* extensions. Functionalities described in these extensions include: non-standard inferences, querying for *told* axioms, retracting *told* axioms, *explanation* services, ABox querying, among others. Now we elaborate on this last extension since we make use of it later on.

ABox Query Interface This interface proposed in [8] extends DIG with the capability to support query answering over the ABox. The query language allows for the expression of Unions of Conjunctive Queries (UCQ) whose basic building blocks are *query atoms* of different types (i.e., concept, role, equality, inequality, and concrete domain types). The interface doesn't predefine specific semantics for query answering, this is left to the implementor's choice. The main element of the query language is the `Retrieve` class, which encompass the query's variables, head and body. Because of space restrictions, we do not provide details here and we refer to the original document for a full description.

3 Managing an ABox via a RDBMS

In this section, we briefly recall the main elements underlying the proposal in [3, 4] for linking existing data stored in a relational database to the instances of the concepts and roles in an ontology. This allows one to effectively manage an ABox via a Relational Database Management System (RDBMS).

An important issue to be taken into account in this context is the notorious problem of *impedance mismatch* between values (data) and objects, which is due to the fact that relational sources store values, whereas instances of concepts are objects denoted by ad hoc identifiers, which are different from data items. To address this problem, [4] proposes to keep data value constants separate from object identifiers, and to consider a domain of object identifiers that are built from data values by applying function symbols. More formally, the alphabet Γ_O of object identifiers is constituted by *value constants* from a set Γ_V, and by *object terms*, i.e., expressions of the form $f(d_1, \ldots, d_n)$, where f is a function symbol of arity n, and $d_1, \ldots, d_n$ are again value constants.

Given Γ_O, we can define an ABox in the standard way as a finite set of membership assertions, in which we may use not only value constants but object terms. To define the semantics of such an ABox, we simply define an interpretation $\mathcal{I}$ in the standard way, and just note that the interpretation function $\cdot^{\mathcal{I}}$ assigns a different element of the interpretation domain to every object identifier in Γ_O (i.e., we enforce the unique name assumption on object identifiers).

We consider now the problem of linking objects in the ontology to the data in a relational database $\mathcal{DB}$. We do so by relying on *mapping* techniques studied extensively in data integration [1]. Specifically, we consider a set $\mathcal{M}$ of *linking axioms*, partitioned into two sets, $\mathcal{M}_t$ and $\mathcal{M}_a$, where:

- $\mathcal{M}_t$ is a set of assertions, called *typing axioms*, each one of the form

$$\Phi \rightsquigarrow T_i$$

  where Φ is a query over $\mathcal{DB}$ denoting the projection of one relation over one of its attributes, and T_i is one of the data types provided by the DL;
- $\mathcal{M}_a$ is a set of assertions, called *mapping axioms*, each one of the form

$$\Phi \rightsquigarrow \Psi$$

  where Φ is a first-order logic query of arity n over $\mathcal{DB}$, and Ψ is a conjunctive query of arity n over the elements of the TBox $\mathcal{T}$, without non-distinguished variables, that possibly includes terms in Γ_O.

We recall from [4] that typing axioms are used to assign appropriate types to constant values appearing in the relations of $\mathcal{DB}$. Basically, these assertions are used for interpreting the values stored in the database in terms of the types used in the ontology. Mapping axioms, on the other hand, are used to map data in the database to concepts, roles, and attributes in the ontology.

It is worth noting that now that we have object terms, the data layer underlying an ontology contains only data, whereas object identifiers are virtually built on top of this data. Thus, autonomous data sources can effectively provide their portion of data and contribute to the ontology instance-level, without being required to agree on any particular object identification scheme.

An *ontology with linking axioms* in a Description Logic $\mathcal{DL}$ is then constituted by a triple $\mathcal{O}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{DB} \rangle$, where

- $\mathcal{T}$ is a TBox expressed in $\mathcal{DL}$,
- $\mathcal{M}$ is a set of linking axioms, and
- $\mathcal{DB}$ is a relational database.

In order to define the semantics of an ontology with linking axioms, we define when an interpretation $\mathcal{I}$ *satisfies a set of linking axioms* $\mathcal{M}$ *w.r.t. a database* $\mathcal{DB}$. Specifically, we say that $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies $\mathcal{M} : \Phi \rightsquigarrow \Psi$ w.r.t. $\mathcal{DB}$ if for each tuple $\boldsymbol{t}$ of value constants in Γ_V, if $\boldsymbol{t} \in ans(\Phi, \mathcal{DB})$[4], then we have that $\boldsymbol{t}^{\mathcal{I}} \in \Psi^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of

[4] We use $ans(\Phi, \mathcal{DB})$ to denote the set of tuples (of the arity of Φ) of value constants returned as the result of the evaluation of the query φ over the database $\mathcal{DB}$.

$\mathcal{O}_m = \langle \mathcal{T}, \mathcal{M}, \mathcal{DB} \rangle$ if and only if $\mathcal{I}$ is a model of $\mathcal{T}$ and satisfies $\mathcal{M}$ w.r.t. $\mathcal{DB}$. With the notion of model in place, we can define all reasoning services over ontologies with linking axioms in the usual way.

Example 1. Consider the following: *(a)* an ontology $\mathcal{O}$ consisting of the axioms: $\mathit{Student} \sqsubseteq \exists \mathsf{Name}$, $\mathit{Student} \sqsubseteq \exists \mathsf{Lastname}$, $\mathit{CumLaudeStudent} \sqsubseteq \mathit{Student}$, where Name and $\mathsf{Lastname}$ are concrete domain roles (i.e., attributes); *(b)* a database $\mathcal{DB}$ and a relation $R \in \mathcal{DB}$ that stores information about students; *(c)* attributes n, l, gpa of R, which store the name, lastname, and gpa of students respectively; *(d)* n, l is a binary key for R; *(e)* n, l are strings. Now we present mapping m_1, which is a mapping axiom, stating how tuples in R are related to *CumLaudeStudent* individuals, and typing axioms m_2, m_3, which *type* the attributes n and l of R w.r.t. the value domains of $\mathcal{O}$:

$$\begin{array}{l}
m_1 : q^1_{db}(n,l) \leftarrow R(n,l,gpa),\ gpa \geq 7.5 \quad \rightsquigarrow \\
\qquad\qquad q_{\mathcal{O}}(\mathbf{stud}(n,l),n,l) \ \leftarrow \ \mathit{CumLaudeStudent}(\mathbf{stud}(n,l)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{Name}(\mathbf{stud}(n,l),n), \mathsf{Lastname}(\mathbf{stud}(n,l),l) \\
m_2 : q^2_{db}(n) \ \leftarrow \ R(n,l,gpa) \quad \rightsquigarrow \quad \mathit{name} \\
m_3 : q^3_{db}(l) \ \leftarrow \ R(n,l,gpa) \quad \rightsquigarrow \quad \mathit{lastname}
\end{array}$$

where each q^i_{db} is a First-Order (i.e., SQL) query expressed over R, $q_{\mathcal{O}}$ is a CQ over $\mathcal{O}$, $\mathbf{stud}(n,l)$ is a function from name-lastname pairs to Γ_O, and name and $\mathit{lastname}$ are the concrete value domains for strings representing names and lastnames, respectively.

4 DIG Extension

We introduce an extension to the *DIG 2.0 Specification* [7] and the *DIG 2.0 ABox Query Interface* [8] that allows for the representation and management of ontologies with *linking axioms*, which are used to populate the ABox with data from different *data sources*. The extension can cover any type of linking assertion and data source by defining a general framework, and allowing the implementor to define specific types of data sources and linking axioms. We also present one implementation of the framework aiming at covering the requirements of linking RDBMS data sources to ontologies based on the work presented in [4].

In the following subsections, we describe the extension by first introducing the general framework. Next, we introduce the implementation of the framework for RDBMS data sources. Finally, we present the extensions to the DIG *request* and *response* classes, which define the interaction with the DIG server. The extension is presented in terms of UML models. One can directly translate the UML models as described in the original DIG 2.0 specification to obtain the corresponding XSD schemas describing the XML based messages [7].

4.1 Ontologies, Data Sources, and Linking Axioms

In Figure 1, we show the classes that compose the framework. These are abstract classes intended to be the base for implementations. Next we elaborate on them.

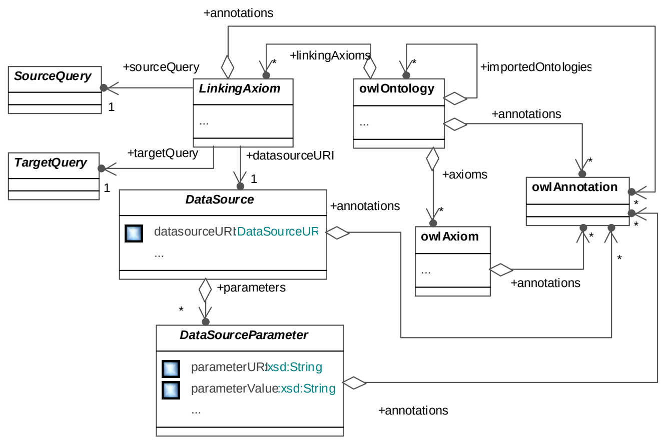

Fig. 1. DIG 2.0 ontology with linking axioms and data sources

A *DataSource* stands for any possible source of data which could be used to populate an ontology's ABox. It is related to a set of *DataSourceParameters*, which provide the information that the DIG server requires to interact (e.g., establish a connection to a server, access a file, etc.) with the given data source. Each data source is identified by a unique *datasourceURI* and each data source parameter is identified by a *parameterURI*. A data source or data source parameter can have any number of *owlAnnotations*, which can be used to attach human readable information. An annotation will have no effect on the interaction of the DIG server with the data source, as is the case with annotations in OWL 1.1.

A *LinkingAxiom* is part of an ontology and is associated to a data source. It indicates the relationship between data in the data source and elements of the ontology. Annotations can be associated also to linking axioms. The main elements of a linking axiom are the *SourceQuery* and the *TargetQuery*. A source query is a specification of how to extract data from the source expressed in some given query language. We do not restrict a priori the query language, which in the most general case could be any computation over the source. Restriction on the query language is done in implementing classes. A target query is a specification of how to extract data over the ontology. As with source queries, there are no restrictions on the language, and implementing classes should define them. A linking axiom states that the data described by the source query is related to the objects described by the target query. The specific way in which this data is related is specified by specific *LinkingAxiom* implementations.

4.2 RDBMS Data Sources, Linking Axioms, and Parameters

Now we present an implementation of the framework whose goal is to allow for the representation of linking axioms of the form presented in [4] (see Figure 2).

An *RDBMSDataSource* stands for any RDBMS accessible to the DIG server. Related to an *RDBMSDataSource* there is a set of parameters. The class *RDBMSPa-*

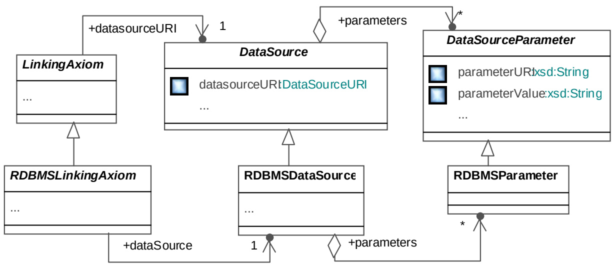

Fig. 2. Implementation of the framework for RDBMS data sources

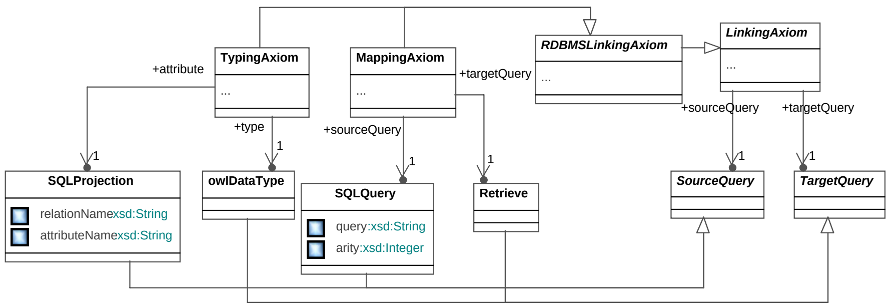

Fig. 3. Linking Axioms

rameter defines a set of URIs of parameters that are always present in any *RDBMS-DataSource*. These URIs are: *RDBMSIP*, *RDBMSUsername*, *RDBMSPassword*, and *RDBMSDatabaseName*. The purpose of the parameters identified by these URIs is self explanatory.

An *RDBMSLinkingAxiom* is related to a RDBMS data source (see Figure 3). We define two types of RDBMS linking axioms, the *TypingAxiom* and the *MappingAxiom*, which correspond to *typing* and *mapping* axioms of Section 3, respectively.

A *MappingAxiom* is a statement of relationship between a query over an RDBMS data source and a query over the DIG ontology. In a mapping axiom, the source query is represented as an *SQLQuery*, which is characterized by a *query* attribute, containing a well formed SQL query, and an *arity* attribute, indicating the arity of the query. The target query is expressed as an UCQ represented as a *Retrieve* object (see Section 2) extended by allowing function symbols in query atoms (see Section 3).

A *TypingAxiom* is a statement of relationship between an attribute of a relation and an OWL data type. In a typing axiom, the source query is represented by an *SQLProjection*, which stands for the projection of the attribute *attributeName* over the relation *relationName*. The target query is represented by an OWL 1.1 data type. The semantics of the typing axiom is as in [4].

ABox Query Interface Extension To allow for the type of mappings mentioned in Section 3, we extend the ABox Query Interface by allowing for the ap-

```
<MappingAxiom>
  <SQLQuery query="SELECT name, lastname
                   FROM R
                   WHERE gpa >= 7.5"
            arity="2" />
  <Retrieve>
    <QueryHead>
      <QueryFunction URI="#stud">
        <QueryVariable URI="#name"/>
        <QueryVariable URI="#lastname"/>
      </QueryFunction>
      <ConcreteDomainQueryVariable
        URI="#name"/>
      <ConcreteDomainQueryVariable
        URI="#lastname"/>
    </QueryHead>
    <QueryBody>
      <QueryObjectIntersectionOf>
        <ConceptQueryAtom>
          <QueryFunction URI="#stud">
            <QueryVariable URI="#name"/>
            <QueryVariable URI="#lastname"/>
          </QueryFunction>
          <owl:OWLClass owl:URI="
             #CumLaudeStudent"/>
        </ConceptQueryAtom>
        <ConcreteDomainQueryAtom>
          <QueryFunction URI="#stud">
            <QueryVariable URI="#name"/>
            <QueryVariable
              URI="#lastname"/>
          </QueryFunction>
          <ConcreteDomainQueryVariable
            URI="#name"/>
          <owl:DataProperty
            owl:URI="#Name"/>
        </ConcreteDomainQueryAtom>
        <ConcreteDomainQueryAtom>
          <QueryFunction URI="#stud">
            <QueryVariable URI="#name"/>
            <QueryVariable
              URI="#lastname"/>
          </QueryFunction>
          <ConcreteDomainQueryVariable
            URI="#lastname"/>
          <owl:DataProperty
            owl:URI="#Lastname"/>
        </ConcreteDomainQueryAtom>
      </QueryObjectIntersectionOf>
    </QueryBody>
  </Retrieve>
</MappingAxiom>
```

Fig. 4. Example of a mapping statement represented as an XML message

pearance of function symbols, in the form of `<QueryFunction>` elements, in `<ConceptQueryAtom>` and `<RoleQueryAtom>` elements.

Example 2. In Figure 4, we present the instantiation of the *MappingAxiom* class for mapping axiom m_1 from Example 1. The axiom is presented in the XML based implementation of *MappingAxiom*. Notice the use of a *function* in query atoms within the ABox Query.

4.3 Managing Data Sources and Linking Axioms

Now we present the extensions to the DIG 2.0 *Requests* and *Responses* that allow for the management of ontologies as presented before. We first present the extensions for data source management (see Figure 5) and later introduce the extensions for linking axiom management (see Figure 6).

The *CreateDataSourceAllocateURI* request instructs the server to create a data source and allocate a unique URI for it. The *CreateDataSource* request tells the DIG server to create a new data source whos URI is *dataSourceURI*. The *ReleaseDataSource* instructs the server to eliminate the given data source and should be used when a data source is not to be accessed by the server anymore. The user is responsible for updating or removing all linking axioms related to the released data source. The *SetDataSourceParameters* takes a set of parameters and adds them to a data source. If a parameter in the set has the same *parameterURI* as one of the parameters already in the data source, the *parameterValue* and annotations of the old parameter should be replaced with the new ones. The DIG server should answer with a *Confirmation* on successful completion of the operations mentioned above. If an error is encountered, a DIG *Error* should be returned describing the issue. The sets of all annotations and parameters

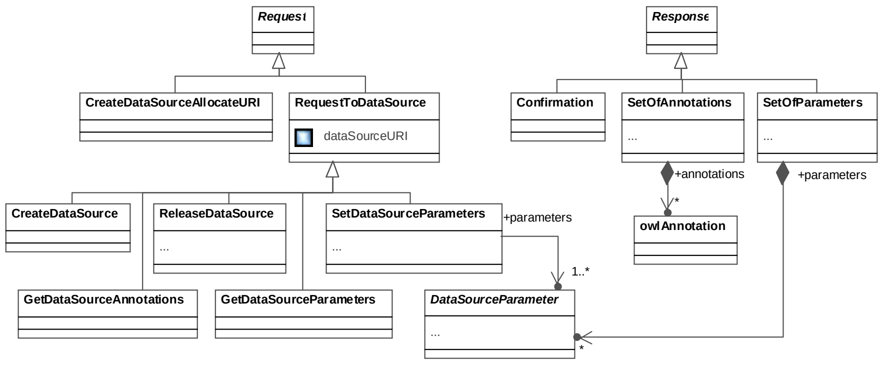

Fig. 5. Data source managements requests

of a given data source can be retrieved using the *GetDataSourceAnnotations* and *GetDataSourceParameters* requests, respectively. To which the server will respond with a *SetOfAnnotations* and *SetOfParameters*, respectively.

Since linking axioms are part of the ontology (see Section 4.1), for the requests extension we use the DIG 2.0 *RequestToOntology* class which refers to the ontology in the *ontologyURI* attribute. The *GetAllLinkingAxioms* request returns the set of all linking axioms in the ontology. A *TellLinkingAxioms* request takes a set of linking axioms and adds them to the ontology. The *RetractLinkingAxioms* takes a set of linking axioms and removes every axiom in the ontology which is structurally equivalent[5] to one of the given axioms. A shortcut for removing all linking axioms in the ontology is given with the *RetractAllLinkingAxioms* request. The server returns a *Confirmation* on successful completion of these requests.

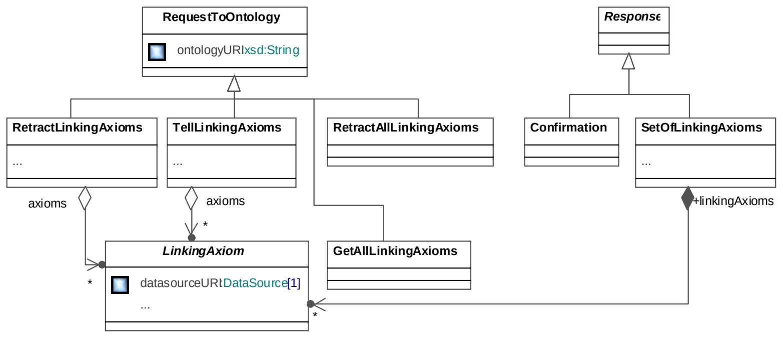

Fig. 6. Linking Axiom Management Requests

[5] Linking axioms are structurally equivalent if they are syntactically the same up to the order of their annotations.

Finally, concerning the behavior of the DIG server w.r.t. the interaction with the data sources, we define the following: (*i*) If it is impossible for the DIG server to interact with a given data source, all source queries of all linking axioms related to that data source should return an empty set and a warning should be issued to the client. (*ii*) If a linking axiom refers to a data source with an unknown URI, the DIG server should respond with an error indicating the issue.

5 Conclusions and Future work

The community working with DL reasoners is increasingly interested in being able to retrieve data from existing data sources and link these data to their ontologies. As the number of DL reasoners supporting such operation grows, the DIG Interface will need to be able to handle this kind of interaction. Aiming at this, we presented an extension of the DIG 2.0 interface that allows a DIG server to offer such a kind of functionality. We present a framework for describing the data sources and the axioms that link them to the ontology. Based on existing work, we introduce an implementation of the framework that can handle RDBMS data sources and linking axioms relating this kind of sources to the ontology using SQL and a form of UCQs. This proposal will be submitted to the Description Logic Implementation group as a starting point for standardization of the requirements described here. Software implementation for the QuOnto system [9] is ongoing.

Acknowledgements. This research has been partially supported by the EU 6th Framework Programme FET project TONES (Thinking ONtologiES), funded under contract FP6-7603.

References

1. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002). (2002) 233–246
2. Heflin, J., Hendler, J.: A portrait of the Semantic Web in action. IEEE Intelligent Systems **16**(2) (2001) 54–59
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic *dl-lite*$_a$. In: Proc. of the 2nd Workshop on OWL: Experiences and Directions (OWLED 2006). (2006)
4. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics (2007) To appear.
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006). (2006) 260–270
6. Bechhofer, S.: The DIG Description Logic interface: DIG/1.0. Technical report, University of Manchester (2002)
7. Bechhofer, S., Motik, B.: DIG 2.0 Specification. Editor's Draft of 02 January 2007. Available at: http://www.cs.man.ac.uk/ bmotik/dig/dig_specification.html
8. Kaplunova, A., Möller, R.: ABox Query Interface proposal. Editor's Draft of 18 January 2007. Available at: http://www.sts.tu-harburg.de/ al.kaplunova/dig-query-interface.html
9. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: QUONTO: QUerying ONTOlogies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005). (2005) 1670–1671