SDQNET: Semantic Distributed Querying in Loosely Coupled Data Sources

Eirini Spyropoulou[1] and Theodore Dalamagas[1]

School of Electr. and Comp. Engineering
National Techn. University of Athens,
Athens, GR 15773
{ispirop,dalamag}@dblab.ece.ntua.gr

Abstract. Web communities involve networks of loosely coupled data sources. Members in those communities should be able to pose queries and gather results from all data sources in the network, where available. At the same time, data sources should have limited restrictions on how to organize their data. If a global schema is not available for such a network, query processing is strongly based on the existence of (hard to maintain) mapping rules between pairs of data sources. If a global schema is available, local schemas of data sources have to follow strict modelling restrictions posed by that schema.
In this paper, we suggest an architecture to provide better support for distributed data management in loosely coupled data sources. In our approach, data sources can maintain diverse schemas. No explicit mapping rules between data sources are needed to facilitate query processing. Data sources can join and leave the network any time, at no cost for the community. We demonstrate our approach, describing SDQNET, a prototype platform to support semantic query processing in loosely coupled data sources.

1 Introduction

Web communities, e.g. concerning e-science, e-learning, art and culture, are popular means of exchanging data and queries for collaborative work. Such communities are based on a network of loosely coupled data sources characterized by heterogeneity and autonomy. A community member in any data source should be able to pose queries and gather results from all the other data sources in the network, where available. At the same time, data sources should have limited restrictions on how to organize their data. Also, a data source should be able to leave/join the network at any time, with no additional global maintenance cost for the community.

Traditional information integration architectures, like virtual databases [9] and mediators [10], provide an arrangement of heterogeneous data sources with a global aspect of the underlying information, independent of its schema and location. However, both architectures are not directly applicable to communities of loosely coupled databases. This is due to their requirement for the existence

of a global mediated schema and mappings between that schema and the (local) schemas of the data sources. A global schema must be prepared carefully and globally, and is usually hard to maintain. Data sources are not autonomous enough to significantly change their schemas. The ad-hoc extensibility of data sources is missing. Thus, even for small-scale loosely coupled data sources, data and query exchanging is difficult to achieve.

Peer-to-peer data management systems (PDMSs) [2] provide an information integration framework closely related to that of loosely coupled databases. PDMSs support decentralized sharing and management of data using data sources that have client and server functionality at the same time. A PDMS consists of a set of data sources. Each one maintains a local schema. Queries are initiated by a data source and propagated to other ones for evaluation.

In case a global schema is available in a PDMS, local schemas are usually views of that global schema. In this case, query routing, i.e. finding which sources are able to answer a query, can be supported by schema indexes built for the local schemas of all sources, like in [1]. Those sources will receive and then process the query. While the existence of a global schema makes query routing and processing easier, it does not provide the necessary flexibility needed to support loosely coupled database communities. The reason is that data sources have to follow strict modelling restrictions imposed by the unique global schema.

In case a global schema is not available, each source maintains a list of neighbouring source. Mapping rules should be provided between a source and its neighbours. Queries initiated by a source are sent to its neighbouring sources. Each one of those sources sends the query to its neighbours, and so on. The mapping rules are used to reformulate the query to match the local schema of each source reached. However, such rules are difficult to maintain. For example, every time that a new source joins the system, new mapping rules should be created and several current mapping rules should be changed manually.

Our approach. In this paper we suggest an architecture to provide better support for distributed data management in loosely coupled data sources. The suggested architecture gives the necessary flexibility to employ diverse schema descriptions in data sources, without the need to maintain mapping rules between data sources.

We demonstrate our approach describing SDQNET, a prototype platform to support semantic query processing in loosely coupled data sources. In SDQNET, schema information is organized in three levels: *local schemas*, *community schemas* and *global schema*. Local schemas (level 1) are the schemas of data sources. Community schemas (level 2) are RDFS schemas available in the community, relevant to a specific domain. They are used to wrap the local schemas of the data sources that want to join the community. The reason for picking-up RDFS is that it can capture easily schema descriptions that range from simple tagged data to relational descriptions and even to complex class/subclass hierarchies. Finally, community schemas are parts of a global RDFS schema (level 3). However, as it will be shown in the next sections, we note that the task of joining a community is based on exploiting the community schemas available and not

the global schema. The global schema just ensures that community schemas are consistent to each other. This gives better flexibility to support loosely coupled database communities and enables query transformation.

To join a community, a data source should agree to exploit schema information from a variety of community schemas and not from only one global schema. Specifically, it should determine an RDFS schema R to wrap its local schema. The R (a) can be part of any available community schema, or (b) can be constructed by applying *schema operators* on the available community schemas. Those schema operators can produce integrated RDFS schemas based on union, intersection and difference semantics.

Once a data source determines an RDFS schema R, it wraps its local schema to R. Wrapping is performed by mapping its local schema primitives to R's primitives (e.g. relational tables to classes, and attributes to properties), and converting local data to RDF resources (e.g. tuples to RDF resources).

Any data source that has joined a community in SDQNET can initiate queries using the RDFS schema which wraps its local schema for that specific community. Queries are initiated and propagated to other data sources in the network to gather results. Query processing does not require the existence of mapping rules between pairs of data sources.

Contribution. The main contribution of this work is an architecture for distributed data management in loosely coupled data sources. The suggested architecture provides the necessary flexibility to support loosely coupled database communities, and it lies between the following two extremes: (a) 'having a global schema for easy query processing, at the expense of the flexibility needed for defining local schemas in nodes' and (b) 'do not have a global schema to give the needed flexibility for defining local schemas in nodes, at the expense of maintaining mapping rules'. Specifically:

- We provide a flexible wrapping mechanism, based on RDFS schemas, for data sources that employ diverse local schema information.
- We exploit schema operators for such wrapping. The operators are applied on RDF schema graphs available for the community, and produce new, integrated ones. Such integration is based on set-like semantics and gives an intuitive way in wrapping data sources.
- We design a query processing technique that does not require the existence of mapping rules during the propagation of the query in the data sources. Under this technique, we are also able to retrieve answers, even in the case a query does not exactly match the schema of a local data source.
- We present SDQNET, a platform that integrates the above ideas to support semantic query processing in loosely coupled data sources.

Related Work. The Piazza system [2] supports decentralized data management in a network of loosely coupled bases. Each node in Piazza can maintain a local schema without any restriction. Query processing is based on reformulating the initial query using mapping rules. In the Edutella [3] system, nodes can either agree to use the same schema or use different schemas. The Edutella Mapping

Service is responsible to handle the mappings between different schemas and use them in order to translate queries from one schema to the other.

However, as we point out in the previous paragraphs, mapping rules are difficult to maintain. When for example a new node joins the network, new mapping rules should be created and several current mapping rules should be changed. Similar problems appear in case a node leaves the network.

The SQPeer system [1] uses the Semantic Overlay Networks (SONs) technology, according to which peers sharing the same schema information about a community domain are clustered together to the same SON. The active schema of the peer is a subset (view) of a unique global SON schema for which all classes and properties are populated in the peer base. The SQPeer system identifies peers that can answer a query by maintaining indexes on schema information in peers. Thus, query processing is not based on query reformulation.

In our approach, data sources do not have to follow strict modelling restrictions imposed by the unique global schema. This is due to the existence of community schemas, and the set of schema operators available which provide the necessary flexibility considering schema selection for the wrapping tasks.

2 Application scenario

In this section we present an application scenario that exploits SDQNET to set up a community network. Based on the scenario, we identify the key features of SDQNET and its functionality to support distributed data management in loosely coupled data sources.

Consider a web community for exchanging information about movies. This community involves a set of data sources and a set of *community RDFS schemas* shown in Figure 1. The oval labelled nodes represent classes. The rectangular labelled nodes denote literals, like string, integer, etc. The plain labelled edges with the solid arrow represent properties. The other arrowed edges define an isA hierarchy (class/subclass) of classes.

To join a community, a new data source should determine, with the help of SDQNET, a schema in order to wrap its local data. In our example, we consider three new data sources. We assume that all data sources maintain relational schemas. Suppose that community schema S_3, shown in Figure 1, fits perfectly the wrapping needs of the first data source DS_1. Thus, DS_1 selects S_3 to wrap its local data.

Suppose now that S_2 fits the wrapping needs of the second data source DS_2, but it also contains schema information not needed in DS_2. For example, DS_2 does not want provide information about action and science fiction movies. So, DS_2 considers only a part of schema S_2, presented as schema S_4 in Figure 2.

Finally, assume that both S_1 and S_3 contain schema information needed by the third data source DS_3. For example, DS_3 provides information not only about actors, but also about producer of movies. Source DS_3 decides to merge S_1 and S_3 to get the appropriate RDFS schema to wrap its data. The merged

schema S_5 is shown in Figure 2. Summarizing, DS_1 maintains schema S_3 to wrap its local data, DS_2 maintains S_4 and DS_3 maintains S_5.

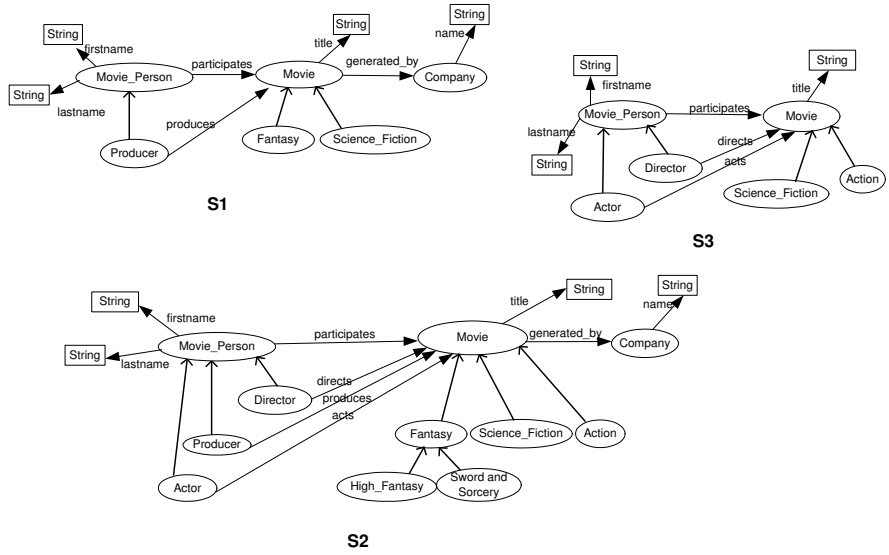

Fig. 1. Community RDFS Schemas.

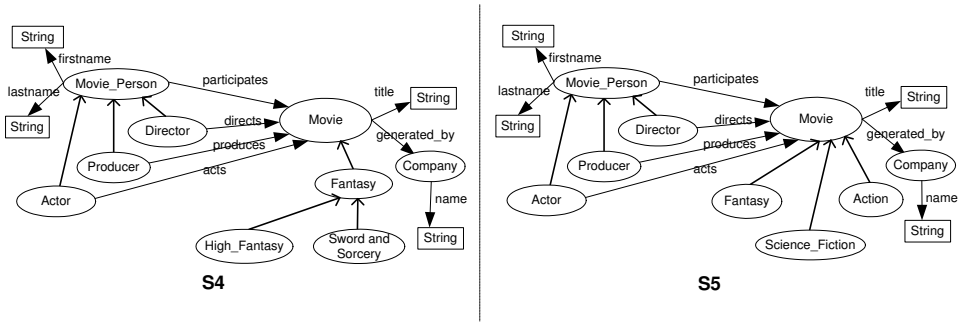

Fig. 2. RDFS schemas that wrap data sources.

Source DS_2 initiates a query to find director names for "High Fantasy" movies generated by "JAK Productions". This query reaches both DS_1 and DS_3. However, DS_1 cannot reply to the answer since it does not contain the class `Company`. Similarly, at first glance, DS_3 can not reply to the answer. However, knowing from the community schemas that "High Fantasy" is a subclass of "Fantasy", we can still get answers, though more general. In fact, the SDQNET gives the option for the user to select whether to get such general answers or not.

Based on the functionality described in the application scenario of this section, we next address the following issues:

1. How a new data source joins the community?
2. How a data source wraps its local schema to community schemas?
3. How a data source initiates a query and how the query is processed?

3 Joining the community

When a data source joins a community network, it determines an RDFS schema R to wrap its local schema. To assist the creation of R, SDQNET provides a set of *community schemas*. Community schemas are RDFS schemas relevant to the domain addressed in the involved community network, and are used to create R. Specifically, R is created by integrating community schemas under union, intersection and difference semantics. Community schemas are RDF schemas which are *subsets* (discussed later) of a given RDF schema, called *global RDF schema* which ensures that community schemas are consistent to each other.

The construction of R is based on the usage of *schema operators* suggested in [6] to support manipulation of RDF schemas as full-fledged objects. The operators are applied on RDF schema graphs and produce new, integrated RDF schema graphs. The key feature is that such integration is based on set-like semantics. We exploit three binary operators (union, intersection, difference) that can be applied on RDF schema graphs as a whole, and produce new ones. We also exploit a unary operator that can be applied on one RDF schema graph and return a part (subset) of it. In the following subsections, we discuss background issues concerning those schema operators originally presented in [6].

3.1 Modelling issues

RDF schemas provide a type system for RDF. The primitives of RDF schemas are classes and properties. Classes describe general concepts and entities. Properties describe the characteristics of classes. They also represent the relationships that exist between classes. Classes and properties are primitives similar to those of the type system of object-oriented programming languages. The difference is that properties in RDF schemas are considered as first-class citizens and are defined independently from classes. Formally, an RDF schema is defined in [6] as follows:

Definition 1. *An RDF schema (RDFS) is a 5-tuple* (C, L, P, SC, SP) *representing a graph, where:*

1. C *is a set of labelled nodes. Each node in* C *represents an RDF class.*
2. L *is a set of nodes labelled with data types defined in XML schema [7], e.g. integer, string etc. Each node in* L *represents a literal.*
3. P *is a set of directed labelled edges* (c_1, c_2, p) *from node* c_1 *to node* c_2 *with label* p*, where* $c_1 \in C$ *and* $c_2 \in C \cup L$*. Each edge in* P *represents an RDF property* p *with domain* c_1 *and range* c_2*.*

4. *SC is a set of directed edges* (c_1, c_2) *from node* c_1 *to node* c_2*, where* $c_1, c_2 \in C$*. Each edge in SC represents an isA relationship between classes* c_1 *and* c_2 *(i.e.* c_1 *is a subclass of* c_2*).*
5. *SP is a set of directed edges* $((c_1, c_2, p_1), (c_3, c_4, p_2))$ *from edge* (c_1, c_2, p_1) *to edge* (c_3, c_4, p_2)*, where* (c_1, c_2, p_1)*,* $(c_3, c_4, p_2) \in P$*. Each edge in SP represents an isA relationship between property* (c_1, c_2, p_1) *and property* (c_3, c_4, p_2) *(i.e. that is* (c_1, c_2, p_1) *is a subproperty of* (c_3, c_4, p_2)*).*

Let $\preceq_C$ be a relation on C: $c_1 \preceq_C c_2$ holds if c_1 is a subclass of c_2. With $\preceq_C^+$ we denote the transitive closure of $\preceq_C$. We consider c_1 to be an *ancestor* of c_2 (or c_2 to be a *descendant* of c_1) if $c_2 \preceq_C^+ c_1$. Similarly, let $\preceq_P$ be a relation on P: $(c_1, c_2, p_1) \preceq_P (c_3, c_4, p_2)$ holds if (c_1, c_2, p_1) is a subproperty of (c_3, c_4, p_2). With $\preceq_P^+$ we denote the transitive closure of $\preceq_P$. We consider (c_1, c_2, p_1) to be an *ancestor* of (c_3, c_4, p_2) (or (c_3, c_4, p_2) to be a *descendant* of (c_1, c_2, p_1)) if $(c_3, c_4, p_2) \preceq_P^+ (c_1, c_2, p_1)$.

We next present the concept of the *subset* relation for RDF schemas introduced in [6]. Intuitively, an RDF schema R_1 is a subset of an RDF schema R_2 when R_1 contains some of the elements (i.e. classes, properties, etc.) of R_2, and it does not violate the *isA* hierarchy of classes and properties maintained in R_2.

Definition 2. *Let* $R_i = (C_i, L_i, P_i, SC_i, SP_i)$ *and* $R_j = (C_j, L_j, P_j, SC_j, SP_j)$ *be two RDF schemas.* R_i *is a subset of* R_j*, denoted by* $R_i \subseteq R_j$*, if:*

1. $C_i \subseteq C_j$.
2. $L_i \subseteq L_j$.
3. *for each edge* $(c_1, c_2, p_1) \in P_i$ *there is an edge* $(c_3, c_4, p_2) \in P_j$ *with* $(c_1 \equiv c_3$ *or* $c_1 \preceq_{C_j}^+ c_3)$ *and* $(c_2 \equiv c_4$ *or* $c_2 \preceq_{C_j}^+ c_4)$ *and* $p_1 = p_2$.
4. *for each pair of nodes* $c_1, c_2 \in C_i$*,* *if* $c_1 \preceq_{C_i} c_2$ *then* $c_1 \preceq_{C_j}^+ c_2$ *and* *if* $c_1 \preceq_{C_j}^+ c_2$ *then* $c_1 \preceq_{C_i}^+ c_2$.
5. *for each pair of edges* $(c_1, c_2, p_1), (c_3, c_4, p_2) \in P_i$*,* *if* $(c_1, c_2, p_1) \preceq_{P_i}^+ (c_3, c_4, p_2)$ *then* $(c_1, c_2, p_1) \preceq_{P_j}^+ (c_3, c_4, p_2)$ *and* *if* $(c_1, c_2, p_1) \preceq_{P_j}^+ (c_3, c_4, p_2)$ *then* $(c_1, c_2, p_1) \preceq_{P_i}^+ (c_3, c_4, p_2)$.

Figure 3 shows the RDF schema R_1 which is a subset of R, since it satisfies all conditions of the definition. For example, having $C_1 = \{A, B, C, E, G\}$ and $C = \{A, B, C, D, E, F, G\}$, $C_1 \subseteq C$. Also, for each pair of nodes in C_1 the fourth condition of the above definition holds (e.g. $A \preceq_{C_1} E$ and $A \preceq_C^+ E$ hold, and $A \preceq_C^+ E$ and $A \preceq_{C_1}^+ E$ hold as well for nodes A, E in C_1).

Finally, we give some definitions which are useful to the discussion that will follow [6]. All subsequent definitions refer to an RDF schema $R = (C, L, P, SC, SP)$.

Definition 3. *The* extended domain *of a property* $(c, s, p) \in P$*, denoted by* $\mathcal{D}^+((c, s, p))$*, is the set of classes* $\{c, c_1, \ldots c_n\}$*, where* $\{c_1, \ldots c_n\}$ *are all descendants of* c*.*

Using the *extended domain* of a property we refer to all classes which can be applied to a property as a set. Similarly, we define the *extended range* of a

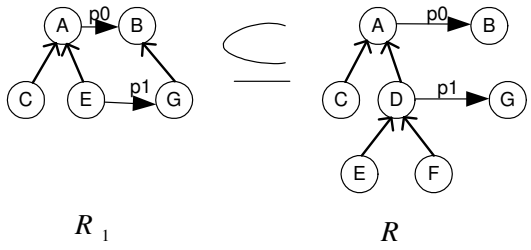

Fig. 3. An example of RDF schema subsets.

property to refer to all the classes from which a property can take values as a set.

Definition 4. *The* extended range *of a property* $(e, c, p) \in P$*, denoted by* $\mathcal{R}^+((e, c, p))$*, is the set of classes* $\{c, c_1, \ldots c_n\}$*, where* $\{c_1, \ldots c_n\}$ *are all descendants of* c*.*

In SDQNET, community schemas are RDF schemas which are subsets of a given RDF schema, called *global RDF schema*. The global schema just ensures that community schemas are consistent to each other.

Definition 5. *Let* $\mathcal{S} = \{R_1, R_2, \ldots R_n\}$ *be a set of RDF schemas. A* global RDF schema *for* S *is an RDF schema* R *such that* $R_i \subseteq R, 1 \leq i \leq n$*.*

3.2 Schema operators

The operators available to construct the RDFS schema R for a data source to join a community network are summarized as follows (a detailed description of the operators is presented in [6]):

1. **Projection**.Given a set of RDF classes, the projection extracts the part of an RDF schema that involves those classes. Consider the RDF schema R in Figure 4. Projecting R with $C_s = \{C, D, G, F\}$ results in an RDF schema which includes classes A, B, C, D, G, F and the involved properties (A, B, p_1) and (D, B, p_2).
2. **Union**.The union operator merges two RDF schemas R_1 and R_2. Union can be implemented as a projection on a class set built from the (set) union of class sets of R_1 and R_2. An example of the union operator is shown in Figure 5.
3. **Intersection**.The intersection operator results in an RDF schema that contains common elements from both schemas. Intersection can be implemented as a projection on a class set built from the (set) intersection of class sets of R_1 and R_2.

4. **Difference**.The difference operator results in an RDF schema that contains elements of one schema that are not present in the other. Difference can be implemented as a projection on a class set built from the (set) difference of class sets of R_1 and R_2.

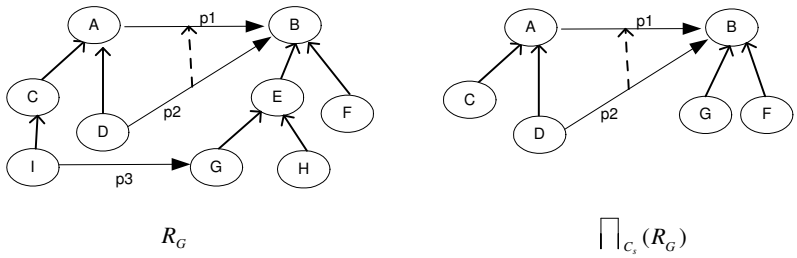

Fig. 4. Projecting R_G with $C_s = \{C, D, G, F\}$.

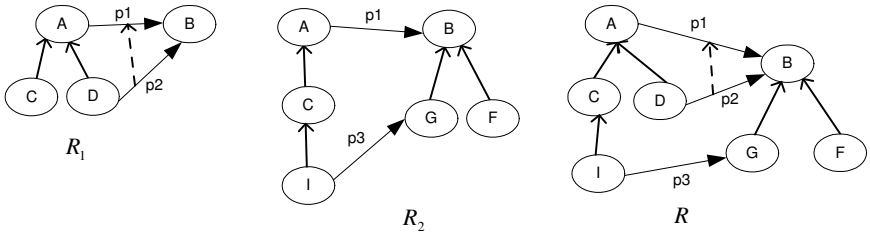

Fig. 5. An example of union operation: $R = R_1 \cup R_2$.

Users in a data source can create an RDFS schema to wrap its local data, and join the community network, using schema creator wizards provided by SDQNET. Thus, the task of joining a community is based on exploiting the community schemas available and not the global schema. The global schema just ensures that community schemas are consistent to each other. This gives better flexibility to support loosely coupled database communities.

When a data source joins the network, it establishes a neighbouring relationship with some data sources (randomly). Such a relationship will be exploited during the query processing phase. (see Section 5).

4 Wrapping local schemas

Next we describe how a data source wraps it local schema to the RDF schema R constructed by the data source as shown in the previous section. In SDQNET, we assume that all data sources employ relational schemas. However, our prototype can be easily extended to handle data sources with schema descriptions that range from simple tagged data to complex class/subclass hierarchies.

The SDQNET performs wrapping based on a mapping between relational model primitives and RDFS model primitives. Users can define such a mapping using wrapper wizards provided by SDQNET. The process includes two steps:

1. First, user defines an SQL view on the relational data available in her data source. Such a view actually determines which data will be offered to the network by the specific data source.
2. Then, the user maps the attributes of the view created in the previous step to certain class properties of the RDFS schema R used by the data source. Those properties have literals as range. Figure 6 shows the wizard provided by SDQNET to define mappings between an SQL view and R on a data source.

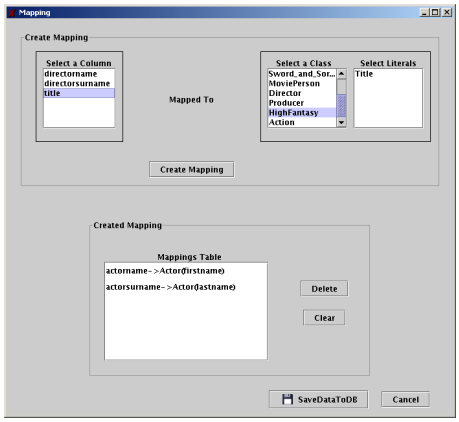

Fig. 6. An example of a mapping between an SQL view and R in a data source.

Formally, a mapping between an SQL view and an RDFS schema R is defined as follows:

Definition 6. *Let* $V(a_1, a_2, \ldots, a_n)$ *be an SQL view for a data source, and* $P = \{(c_1, l_1, p_1), (c_2, l_2, p_2), \ldots, (c_k, l_k, p_k)\}$ *a set of properties of an RDFS schema* (C, L, P, SC, SP)*, where* $c_1, c_2, \ldots, c_k \in C$ *and* $l_1, l_2, \ldots, l_k \in L$*. A mapping* $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{P}$ *is a set of correspondences of the form* $a \sim p$*, where* $a \in \{a_1, a_2, \ldots, a_n\}$ *and* $p \in \{p_1, p_2, \ldots, p_k\}$*.*

For example, in Figure 6, the mapping between the SQL view and the RDFS schema is $\{(actorname \sim firstname), (actorsurname \sim lastname)\}$.

After a mapping is established between the SQL view and the RDFS schema R of the data source, data included in that view are converted to RDF resources. Specifically, all tuples in the view are transformed to RDF resources.

4.1 SQL view-to-RDF conversion

The SQL view-to-RDF conversion is based on the mapping between the SQL view and the RDFS schema. The result of the conversion is an XML encoded RDF file F [8] which wraps the data of the SQL view. More specifically, we use (a) the RDFS schema of the data source to define the structure of F, and (b) the mapping to fill the values of the RDF class properties that appear in F with the values from the corresponding SQL view attributes. Next we present the conversion algorithm in detail.

Algorithm
Consider:
$R = (C, L, P, SC, SP)$: RDFS schema to wrap local schema,
$V(a_1, a_2, \ldots, a_n)$: SQL view,
$P' = \{(c_1, l_1, p_1), (c_2, l_2, p_2), \ldots, (c_k, l_k, p_k)\}$
where $C' = \{c_1, \ldots, c_k\} \subseteq C$ and $L' = \{l_1, \ldots, l_k\} \subseteq L$,
$\mathcal{M}$ = the set of mappings $a \sim p$,
$ClassesToWrite$: it keeps the start tags for RDFS classes to be written (vector, initially empty)
$WrittenClasses$: it keeps the end tags for RDFS classes to be written (vector, initially empty)
$ClosedClasses$: it keeps the end tags for RDFS classes already been written (set, initially empty)
$ClosedProperties$: it keeps the end tags for RDFS properties already been written (set, initially empty)

```
1 for every tuple of V
2   for every class c_i ∈ C'
3     put c_i in ClassesToWrite
4     while ClassesToWrite is not empty
5       if the RDF/XML file is empty
6         write the class c_i start tag
7       else
8         find the property p ∈ P such that (d, c_i, p), where d ∈ C'
9         write the start tag of the property p
10        write the start tag of the class c_i
11      end if
12      for every p_i ∈ P'' ⊆ P', P'' = {(c_i, l_1, p_1), (c_i, l_2, p_2), ..., (c_i, l_k, p_k)}
13        find a such that a ∼ p_i and its value from V
14        write at the RDF/XML file p_i and the value of a
15      end for
```

```
16      put c_i in WrittenClasses
17      remove c_i from ClassesToWrite
18      for every p_i such that (c_i, r, p_i), r ∈ C'
19       put r at the beginning of the ClassesToWrite
20      if there isn't any p_i such that (c_i, r, p_i), r ∈ C'
21       for every wc_i in WrittenClasses, beginning with the last one
22        for every p_i such that (wc_i, r, p_i), r ∈ C'
23          if r ∈ ClosedClasses
24           write the property end tag
25           put p in ClosedProperties
26          end if
27        end for
28        if there is no p_i such that (c_i, r, p_i), r ∈ C' and p_i ∉ ClosedProperties
29          write the end tag of the class wc_i
30          put wc_i in ClosedClasses
31        end if
32       end for
33      end if
34     end while
35  end for
36 end for
```

For example, consider that a user of the data source DS_1 (see Section 2) wants to wrap its local data using S_3 (see Figure 1). First, the user defines the SQL view shown in Table 1. Then, she defines the mapping: $Name \sim firstname$, $Surname \sim lastname$, $Movie \sim title$ ($firstname$, $lastname$ and $title$ are properties whose domain are classes $Actor$, $Actor$ and $ScienceFiction$ respectively). The resulting RDF file in XML encoding follows:

```
<sref:Actor rdf:about="#Item0">
  <sref:firstname>Jodie</sref:firstname>
  <sref:lastname>Foster</sref:lastname>
  <sref:acts>
    <sref:ScienceFiction rdf:about="#Item1">
      <sref:title>Contact</sref:title>
    </sref:ScienceFiction>
  </sref:acts>
  ...
</sref:Actor>
```

Name	**Surname**	**MovieTitle**
Jodie	Foster	Contact
...	...	...

Table 1. SQL View

We first write the start tag of the class *Actor* (lines 5,6). Then, we write the properties of *Actor* which have literals as range and their values according to the mapping (lines 12-14). The class *ScienceFiction* is the only class contained in the $ClassesToWrite$ vector (lines 18,19). So, this is the next class we consider. This class belongs to the range of *acts* property. So, we first write the start tag of *acts* and then the start tag of *ScienceFiction* (lines 8-10). Next, we write the properties of *ScienceFiction* which have literals as range. Since *ScienceFiction* has not any properties which have a class as range, we start closing the tags at the reverse order we opened them (lines 20-32).

In SDQNET, we maintain RDF resources and RDFS schemas for data sources using the ICS-FORTH RDFSuite[1].

5 Querying

A query is initiated by a data source and propagated to its neighbours in the community network for evaluation. The neighbours propagate the query to their own neighbours and so on. We next describe how queries are formulated and processed.

The queries are initiated in a data source using the SDQNET query wizard. Figure 7 illustrates an example of a query that searches for movie titles as well as for the names of their directors.

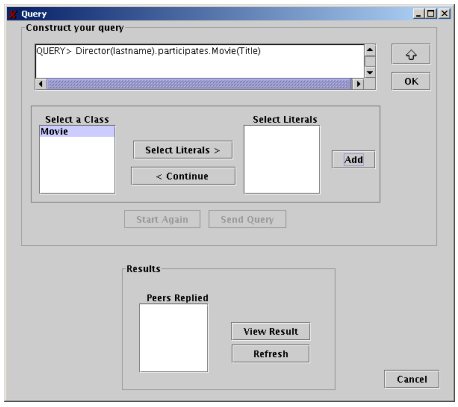

Fig. 7. A query example in SDQNET.

Definition 7. *Let an RDF schema* $R = (C, L, P, SC, SP)$. *A query* Q *on* R *is formed as* $\{\{c_1(P_1), p_1, c_2(P_2), p_2, \ldots, p_{n-1}, c_n(P_n)\}, \mathcal{C}\}$ *where:*

1. $c_i \in C$ *and* $p_i \in P$ $(1 \leq i \leq n)$,

[1] http://www.ics.forth.gr/isl/RDF/index.html

2. *p_k is the domain of c_k and has c_{k+1} as range ($1 \leq k \leq n-1$),*
3. *P_i is a list of properties $\in P$ that have c_i as domain and literals as range ($1 \leq i \leq n$),*
4. *$\mathcal{C}$ is set of conditions of the form $c.p\{=, >, <, <>\}$constant, where p is a property that has $c \in C$ as domain and a literal as range.*

Note that $\{c_1(P_1), p_1, c_2(P_2), p_2, \ldots, p_{n-1}, c_n(P_n)\}$ (from now on: query schema) is actually an RDFS schema, and, in particular, it is a subgraph of R.

A query q matches an RDFS schema R if the RDFS graph that corresponds to $\{\{c_1(P_1), p_1, c_2(P_2), p_2, \ldots, p_{n-1}, c_n(P_n)\}$ (i.e., query schema) is subset of R (see Definition 2). In this case we say that q is satisfiable.

If the query is satisfiable on the RDFS schema R that wraps a data source DS, then it is evaluated on DS. Query evaluation is done using RQL [4], a query language for RDF bases. If the query is not satisfiable then SDQNET applies a query transformation algorithm exploiting semantic information. We next describe the algorithm.

Query transformation algorithm
Consider:
The data source RDFS schema $R = \{DC, DL, DP, DS, PS\}$
The query schema $QS = \{QC, QL, QP, \emptyset, \emptyset\}$
The global RDFS schema $GS = \{GC, GL, GP, SC, SP\}$
$\mathcal{R}^+$: The extended range of a property

1 if $c_1 \notin DC$
2 find the nearest superclass sc of c_1 with $c_1 \preceq^+_{GC} sc$
such that $sc \in GC$ and $sc \in DC$
3 if sc exists and $LP' \subseteq LP$ with $LP = \{lp \in LP | (sc, l, lp)$ and $l \in DL\}$
and $LP' = \{lp' \in LP' | (c_1, l', lp')$ and $l' \in QL\}$
4 substitute c_1 with sc in the query schema
5 end if
6 if $p_1 \notin DP$
7 find the property sp with (sc, c, sp), $c \in GC$ and $sp \in GP$
such that $sp \in (p_1 \cup P' : \forall p' \in P', p_1 \preceq^+_{GP} p')$
8 if sp exists then substitute p_1 in the query schema with sp
9 end if
10 end if
11 for every $c_i \in QC$ and every $p_i \in QP$, $i \geq 2$ do
12 if $c_i \notin DC$
13 find the nearest superclass r of c_i with $c_i \preceq^+_{GC} r$
such that $r \in \mathcal{R}^+$ of p_{i-1} and $r \in DC$
14 if r exists and $LP' \subseteq LP$ with $LP = \{lp \in LP | (r, l, lp)$ and $l \in DL\}$
and $LP' = \{lp' \in LP' | (c_i, l', lp')$ and $l' \in QL\}$
15 substitute c_i with r
16 end if
17 if $p_1 \notin DP$
18 find the property sp with (c_i, c, sp), $c \in GC$ and $sp \in GP$
such that $sp \in (p_i \cup P' : \forall p' \in P', p_i \preceq^+_{GP} p')$
19 if sp exists then substitute p_i in query schema with sp

20 end if
21 end if
22 end for

For example, consider the data source DS_1 of the application scenario in Section 2, and the query q:
Director(firstname,lastname).directs.High_Fantasy.generated_by.Company(name), Company.name="JAK Productions".
DS_1 maintains schema S_3 to wrap its local data (see Figure 1). The query is not satisfiable in DS_1 because the query schema (QS) is not subset of S_3. Also, q cannot be transformed because there isn't any superclass of *Company* (line 13).

Moreover, the query q is not satisfiable in DS_3 because the query schema (QS) is not subset of S_5 (DS_3 maintains schema S_5 to wrap its local data - see Figure 2). According to the transformation algorithm $Director \in DC = \{Actor, Producer, Director, MoviePerson, Fantasy, Science\_Fiction, Action, Movie, Company\}$ (line 1) so we next consider *High_Fantasy* class (line 11). $High\_Fantasy \notin DC$ (line 12). We now search for the nearest superclass of *High_Fantasy* that belongs to the $\mathcal{R}^+$ of *directs* (line 13). The class *Fantasy* satisfies these criteria. In addition, class *High_Fantasy* in QS does not have any properties with literals as range, so $LP' \subseteq LP = \{\text{title}\}$ (line 14). Thus, class *Fantasy* is substituted by class *High_Fantasy* in q (line 15). The property *generated_by* exists in $DP = \{participates, directs, produces, acts, generated\_by\}$ (line 17). We now consider class *Company*. $Company \in DC$ (line 12), so the algorithm terminates and no other substitution is performed. The reformulated query is therefore:
Director(firstname,lastname).directs.Fantasy.generated_by.Company(name), Company.name="JAK Productions".
This new query is satisfiable in DS_3, since the new query schema is subset of S_5.

6 Conclusion

We presented an architecture for distributed data management in community networks of loosely coupled data sources. The suggested architecture is flexible enough for such kind of networks, since it lies between the following two extremes: (a) 'having a global schema for easy query processing, at the expense of the flexibility needed for defining local schemas in nodes' and (b) 'do not have a global schema to give the needed flexibility for defining local schemas in nodes, at the expense of maintaining mapping rules'. No explicit mapping rules between data sources are needed to facilitate query processing. Data sources can join and leave the network any time, at no cost for the community.

Our ideas are being implemented in SDQNET, a platform that supports semantic query processing in loosely coupled data sources. We exploit schema operators applied on RDF schema graphs available for the community. The new, integrated schemas produced are used to wrap data sources. We described a

wrapping mechanism based on RDFS schemas for data sources that employ diverse local schema information. We presented a query processing technique that does not require the existence of mapping rules during the propagation of the query in the data sources. Under this technique, we are also able to retrieve answers, even in the case a query does not exactly match the schema of a local data source.

For further work, we plan to extend our wrapping engine to support data sources that maintain different types organization (e.g., DTDs, flat files, etc). Also, we are working on exploiting SDQNET to set up a real community network to provide art information involving various diverse data sources.

References

1. Giorgos Kokkinidis and Vassilis Christophides. *Semantic Query Routing and Processing in P2P Database Systems: The ICS-FORTH SQPeer Middleware* . In *Proc. of the P2P DB'04 International Workshop, Heraklion, Greece*, 2004.
2. Alon Y. Halevy, Zachary G. Ives, Peter Mork and Igor Tatarinov. *Piazza: Data Management Infrastructure for Semantic Web Applications.* In *Proc. of the WWW'03 International Conference, Budapest, Hungary*, 2003.
3. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjorn Naeve, Mikael Nilsson, Matthias Palmer and Tore Risch. *EDUTELLA: a P2P Networking Infrastructure based on RDF* . In *Proc. of the WWW'02 International Conference, Honolulu, Hawaii, USA*, 2002.
4. Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis and Michel Scholl. *RQL: A Declarative Query Language for RDF.* In *Proc. of the WWW'02 International Conference, Honolulu, Hawaii, USA*, 2002.
5. Sofia Alexaki, Vassilis Cristophides, Gregory Karvounarakis, Dimitris Plexousakis and Karsten Tolle. *The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases.* In *Proc. of the SemWeb'01 International Workshop, Hong-Kong*, 2001.
6. Zoi Kaoudi, Theodore Dalamagas and Timos Sellis. *RDFSculpt: Managing RDF Schemas under Set-like Semantics.* In *Proc. of the ESWC'05 International Conference, Heraklion, Greece*, 2005.
7. W3C Recommendation. XML Schema Part 2: Datatypes Second Edition, 2004. http://www.w3.org/TR/xmlschema-2/.
8. W3C Recommendation. RDF Primer, 2004. http://www.w3.org/TR/rdf-primer/.
9. Amihai Motro. *Superviews: Virtual Integration of Multiple Databases.* In *IEEE Transactions on Software Engineering, 13(7)*, 1987.
10. Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey Ullman, Vasilis Vassalos and Jennifer Widom. *The TSIMMIS approach to mediation: Data models and Languages.* In *Journal of Intelligent Information Systems*, 1997.