Fourth International Conference on Semantics, Knowledge and Grid

P2P Schema-Mapping over Network-bound XML Data

Carmela Comito [1], Domenico Talia [2]

DEIS - University of Calabria
Via P. Bucci 41 c,87036, Rende, Italy
[1]ccomito@deis.unical.it
[2]talia@deis.unical.it

Abstract— **The rise in availability of web-based data sources has led to new challenges in data integration systems for obtaining decentralized, wide-scale sharing of data preserving semantics. In this paper we present a framework for integrating heterogeneous XML data sources distributed over a large-scale, highly dynamic network of autonomous nodes. We propose a query reformulation algorithm to combine and query distributed XML databases through a decentralized point-to-point mediation process among the different data sources by using P2P schema mappings. More precisely, our integration model is based on path-to-path mappings, using the XPath language. We demonstrate the utility and scalability of our ideas and algorithms with a detailed set of experiments. Finally, we present our experience implementing the above cited query reformulation algorithm as a Web service within the GDIS system, a service-based Grid architecture. We have evaluated GDIS on several real world schemas with promising results.**

I. Introduction

The continuously growing availability of data sources over network-bound systems, requires a coordinated and integrated access of such data due to the heterogeneity of the involved data models. The scale, decentralization and dynamism involved, make traditional centralized approaches to information management fully inadequate as they fail to respond to the extreme dynamism of such systems. Decentralized-distributed data structures have recently received a lot of attention with the successful introduction of peer-to-peer systems, Web services, Grids, and ubiquitous computing systems. However, such systems lack an integrated and efficient approach to semantic data sharing. Therefore, semantic interoperability is a key issue to be addressed in open networked systems where many different and independent enterprise parties need to cooperate and share information resources.

In order to address the semantic interoperability concern, in this paper we focus on recovering *schema heterogeneity* presenting an extended and modified version of the *XML* Schema *MAP*ping (*XMAP*) framework [1]. By designing this framework, we aim at developing a decentralized network of semantically related XML schemas that enables the formulation of queries over distributed, heterogeneous data sources. The environment is modeled as a system composed of a number of geographically distributed nodes, where each node can hold one or more XML databases. These nodes are connected to each other through declarative mappings rules.

We recover schema heterogeneity by mapping different schemas following the peer-to-peer (P2P) integration approach recently adopted in the database community. This approach is not based on a global schema but each database (peer) represents an autonomous information system, and data integration is achieved by establishing mappings directly among the various peers. However, differently from related works XMAP does not require heavyweight mapping creation from the user who has only to establish simple correspondences among paths in different schemas and the system supplies the rest. In particular, XMAP uses an algorithm which automatically determine rewritings of the user query, from the correspondences. XMAP is inspired to Piazza [2], but whereas schema mapping in XMAP is based on path-path mappings, in Piazza is based on schema-to-schema mappings. Moreover, in XMAP mappings are expressed as path expressions and the reformulation algorithm allows only for XPath query. Conversely, in Piazza mappings are described as query expressions using a subset of XQuery. Differently from XMAP, the PEPSINT [3] approach is built on a hybrid super-peer architecture in which schema heterogeneity is recovered through a P2P schema-based formalism, whereas data heterogeneity is resolved through a global RDF ontology. The EDUTELLA [4] approach is based on RDF and like XMAP, it relies on a mapping network between local schemas that allows building new mappings transitively. However, differently from EDUTELLA our approach is completely decentralized in the sense that it does not rely on super-peers. Hyperion [5] combines mapping tables that relate different values (as opposed to paths in the present work) across peers, and mapping expression (analogously to XMAP). While semantically impoverished, we use simple element correspondences for two main reasons. First of all, our mappings are to be considered in a large-scale framework and we do not expect a database administrator/user to know the rule machinery, whereas it is reasonable to assume that even users unfamiliar with the complex structure of the schema can provide such correspondences. Hence, the design was motivated by practical considerations. In addition, automated techniques for schema matching (including CUPID, LSD and DIKE) have proven to be very successful in extracting such correspondences.

We have embedded XMAP in the GDIS [6], [7], [8] architecture in which the XMAP algorithm is exposed as a web service [9] within an OGSA-Grid infrastructure. In this system, data is exchanged through invocations of OGSA-DAI web services. We have also evaluated GDIS on several real world schemas with promising results.

The remainder of the paper is organized as follows. Section II presents the XMAP integration framework describing both the mapping approach and the query reformulation process. The experimental evaluation of the XMAP is detailed in section III. Section IV illustrates the deployment and experiencing of the XMAP framework on a service-based Grid architecture. Finally, Section V draws some conclusions.

II. The XMAP Framework

In this section, we describe a modified and extended version of the XMAP framework whose preliminary release has been presented in [1]. Compared to the original work, we now modified the reformulation algorithm by distributing the whole reformulation process, as will be detailed in sub-section II-B. Further, while in [1] we just presented the logical model of the framework and the theoretical grounds supporting it, in this paper we also discuss the experimental evaluation (see subsection III) of a real software prototype of the whole XMAP framework implemented in Java.

A. P2P Schema-Mapping in XMAP

XMAP [1] is a decentralized network of semantically related XML schemas that enables the formulation of queries over heterogeneous, distributed, highly dynamic XML databases. The main goal of the XMAP framework is to allow transparent access to heterogeneous XML data independently of where data is stored in a *large-scale*, *highly dynamic*, *distributed* environment. The main challenge here is heterogeneity of data representations, also known as the problem of schema heterogeneity. There may be considerable differences in the way the sources organize their data, including differences in data representations (terminological), as well as differences in underlying schemas (structural). We recover schema heterogeneity considering the *schema mapping problem*, where given two separate schemas it is necessary to translate data from one to the other. We use a simple form of correspondence: element (attribute) correspondences also specifying the logical access paths that define the associations between elements involved. We require the database administrator or the final user to supply only very simple correspondences. These correspondences can either be created by hand or through some (semi-)automatic mapping discovery algorithm. From such correspondences we specify mappings as path expressions that relate a specific element or attribute (together with its path) in the source schema to related elements or attributes in the destination schema. The data integration model we propose is indeed based on path-to-path mappings expressed in the XPath [10] query language, assuming XML Schema as the data model for XML sources. Specifically, this means that a path in a source is described in terms of XPath expressions. As a first step, we consider only a subset of the full XPath language. The expressions of such a fragment of XPath are given by the following grammar:

$$q \rightarrow n \mid . \mid q \ / \ q \mid q \ // \ q \mid q \ [\ q \ ]$$

where "*n*" is any label (node tests), "." denotes the "current node", "/" indicates the child axis whereas "//" the descendant axis, and "[ ]" denotes a predicate.

A schema mapping is defined as a set of “formulas” that relate a pair of schemas. More precisely, we define a mapping M over a source schema S as a set of mapping rules $\mathcal{R}^M = \{R_1^M, R_2^M, \ldots R_k^M\}$. As we perform path-to-path mappings, a mapping rule associates paths in different schemas. Specifically, a mapping rule is an expression of the form:

$$R^M : \{S_S, P_S\} \longrightarrow_{C^M} \{S_D, P_D\}, \text{ where:}$$

R^M is the label of the rule; S_S is the source schema with respect to which the rules are established; P_S is a path expression in the source schema; S_D is the target schema with respect to which the semantic connections are established; P_D is a path expression in the destination schema (the cardinality of this element may be more than one); C^M denotes the cardinality of the mappings between the two schemas. Mappings are classified as 1-1, 1-N, N-1, N-N according to the number of nodes (both elements and attributes) of the schemas involved in the mapping relationship. The mapping rules are specified in XML documents called XMAP documents. Each source schema in the framework is associated to an XMAP document containing all the mapping rules related to it.

The P2P paradigm has been recently adopted in the database community to overcome the limitations of distributed database systems, namely the static topology and the heavy administration work, and to exploit the dissemination of data sources over the Internet. Accordingly, XMAP is inspired to such trends emerged in the context of peer-to-peer data integration (e.g., [11], [4], [5], [2], [3]). As such, in XMAP each data source represents an autonomous information system, and schema heterogeneity is handled by establishing mappings directly among the various source schemas without resorting to any hierarchical structure. Therefore, in our model, there is no global schema representing all data sources in a unique data model but a collection of local schemas (the native schema of each data source). Regardless of the total number of nodes composing the system, each source schema is directly connected only to a small number of other schemas. However, it remains reachable from all other schemas that belong to its “transitive closure”. For any mapping M, its closure is defined as the set of rules that can be derived from M by repeated composition of schema paths. In other words, the system supports two different kinds of mapping to connect schemas semantically: *point-to-point* mappings and *transitive* mappings. In transitive mappings, data sources are related through one or more “mediator schemas”. For example, if we have a source A directly connected to a source B and B connected to C, A is connected to *both* B and C. Establishing the mappings this way creates a graph of semantically related sources where each of the sources knows its direct semantic neighbors (point-to-point mapping) and can learn about the mappings of its neighbors (transitive mapping).

The XMAP framework abstracts from the underlying network infrastructure, it is modeled as a number of various autonomous nodes (that can be also referred to as sites, sources, peers, etc) which hold information, and which are linked to other nodes by means of mappings. Therefore, it can be extended at information nodes in any networked environment, and, as thus it can be seen as a set of network nodes connected to the Internet. More precisely, XMAP is composed of a collection $\mathcal{N}$ of *nodes* which are logically bound to XML data sources. That is, each data source D_n is represented by exactly one node n and, conversely, each node has access to a single data source, named *local data source*. Naturally, a *local schema* S_n is associated to this data source D_n. Data sources employ the XML data model and each source defines its own XML Schema. Each node also holds a collection of mappings M_n from its local schema to other foreign schemas. Finally, a node knows a list (also named *partial view* or, simply, *view*) of other nodes (called *neighbors*). These nodes are connected to each other through declarative mappings rules. Finally, we make an "open world" assumption: nodes do not have complete knowledge of the domain but every node can contribute new answers to the user queries.

B. Query Reformulation

Answering a query in XMAP is done by *reformulating* it over the schemas of the semantically-connected nodes in the framework. Precisely, our query processing approach exploits the semantic connections established in the system by performing the *XPath query reformulation algorithm* before executing the input query, in order to gain further knowledge. This way, when a query is posed over the schema of a source, the system will be able to use data from any source that is transitively connected by semantic mappings. Indeed, it will reformulate the given query expanding and translating it into appropriate queries for each semantically related source. In this way, the appropriate query will be posed on that node, and additional answers may be found. Thus, the user can retrieve data from all the related sources in the system by simply submitting a single XPath query. The result of query reformulation is a union of reformulated queries: one or more queries per node schema.

In the original XMAP reformulation algorithm [1] we assumed that the node receiving the query submitted by the client is the one that performs the full (transitive) reformulation. Here nodes may exchange both data and mappings, so that only the query node will eventually evaluate the query answer in one go. Therefore, there is no distributed computation and the network may be flooded with data. This is the simplest way to detect and eliminate redundant queries. However, this also imposes a high load on the contacted node and triggers a heavy-weight and complex management of mapping information that must be kept consistent across all nodes in the transitive closure. To leverage these limitations, in this paper we introduce a variation of the query processing algorithm where the reformulation process is fully decentralized and distributed. Differently from the original design, here a node applies only one step of the reformulation algorithm to produce only its direct reformulations over the schemas directly connected to its schema. In this case, nodes only need to know mappings having their local schema as the source schema. Therefore, the algorithm is composed of several reformulation steps, and each of such steps performs direct reformulations by using the point-to-point mappings. To obtain transitive reformulations of a query it is necessary to concatenate individual reformulation steps by exploiting all the mappings in the transitive closure of the schema over which the original query is formulated. Each time a reformulated query is obtained, the algorithm tries to rewrite it by recursively invoking the XMAP algorithm.

The algorithm can be decomposed in the following stages (see Figure 1):

1) *Identifying the path expressions* in Q.
   An XPath query can contain one or more predicates that produce different branching points in the tree pattern representing the query. Each of these branches identifies a specific path in the XML data source. The paths identified in the query are collected into a set $\mathcal{P}$.
2) *Looking for corresponding paths in all source schemas related to S.*
   The goal of this stage is to find corresponding paths in all sources semantically related to S. This means finding the path expressions corresponding to every element P_i in $\mathcal{P}$, by using the mapping information specified in the XMAP document provided with S. These paths $P^{\diamond}_{i,j}$ are called *corresponding paths*, and the schema $S^{\diamond}_j$ they belong to, *corresponding schema*. In particular we define a *corresponding element* $E^{\diamond}_{i,j}$ as a tuple $\langle S^{\diamond}_j, \{P^{\diamond}_{i,j}\}\rangle$, where $\{P^{\diamond}_{i,j}\}$ is a set of paths over the schema $S^{\diamond}_j$. A *corresponding set* $\mathcal{E}^{\diamond}$ is a set of corresponding elements $\{E^{\diamond}_1, \dots E^{\diamond}_n\}$ (with $E^{\diamond}_j = \bigcup_i E^{\diamond}_{i,j}$).
3) *Pruning of corresponding schemas.*
   The third stage of the algorithm checks for each corresponding schema found in the previous stage whether it may be used to obtain one or more reformulation of the query Q. To this aim, the algorithm checks whether each of such schemas has at least one corresponding path for each path present in the query. The schemas that meet this condition are the only ones that we will be considered to obtain reformulated queries, we call them candidate schemas. We define a *candidate element* $E^{\star}_{i,j}$ as a tuple $\langle S^{\star}_j, \{P^{\star}_{i,j}\}\rangle$, where $\{P^{\star}_{i,j}\}$ is a set of paths over the schema $S^{\star}_j$. A *candidate set* $\mathcal{E}^{\star}$ is a set of candidate elements $\{E^{\star}_1, \dots E^{\star}_n\}$ (with $E^{\star}_j = \bigcup_i E^{\star}_{i,j}$).
4) *Constructing reformulated queries.*
   In this stage, given the set $\mathcal{E}^{\star}$, the algorithm produces one or more XPath queries over each schema in the set. More precisely, for each destination schema $S^{\star}_j$ in $\mathcal{E}^{\star}$ the following steps are performed:

a) *Checking Join conditions.* Once the cardinality of the mapping has been established, and before actually producing the query, one needs to check the join conditions between the paths $P^\star_{i,j}(1 \leq i \leq |\mathcal{P}|)$ of the candidate schema $S^\star_j$. So, this step produces as output a set $\mathcal{E}^{CR}$ of *candidate reformulation element* $E^{CR}_{i,j}$.
b) *Pruning redundant reformulations.* If a candidate reformulation element in the set $\mathcal{E}^{CR}$, found in the previous stage, has already been used to reformulate the original query, it will not be used again to reformulate the same query. The output of this sub-stage is the set $\mathcal{E}^{CR}$ pruned of the candidate reformulation elements already been used.
c) *Composing XPath Queries.* Once the reformulation set has been obtained, the actual production of one or more XPath queries is initiated. These queries are the product of the reformulation of the query Q in the destination schema $S^\star_j$. The step produces as output the set $\mathcal{Q}^\star$.

At this point all the direct reformulations of the original query have been produced. Then, the algorithm is recursively invoked over them by exploiting transitive mappings.

```
Algorithm QueryReformulation
Input: query $Q$, schema $S$, mapping $M$ ($M$ is the XMAP of $S$)
Output: set of reformulated queries $\mathcal{Q}^\star$

begin
    $\mathcal{P} \leftarrow$ IdentifyPath($Q$);
    for each path $P_i \in \mathcal{P}$ do
        $\mathcal{E}^\diamond \leftarrow$ FindCorrespondingPath($P_i$, $M$);
    $\mathcal{E}^\star \leftarrow$ PruningSchema($\mathcal{E}^\diamond$);
    for each $S^\star_j \in \mathcal{E}^\star$ do
        if ($|\mathcal{P}|$=1) then
            for each candidate element $E^\star_{i,j} \in \mathcal{E}^\star$ do
                $E^{CR}_{i,j} \leftarrow$ ConstructCandidateReformulations($E^\star_{i,j}$)
            for each candidate reformulation $E^{CR}_{i,j} \in \mathcal{E}^{CR}$ do
                if ( ! RedundantQuery($E^{CR}_{i,j}$)) then
                    $Q^\star \leftarrow$ ConstructQuery($E^{CR}_{i,j}$);
        else
            $\mathcal{E}^{CR} \leftarrow$ CombinePaths($E^\star_{i,j}$);
            for each candidate reformulation $E^{CR}_{i,j} \in \mathcal{E}^{CR}$ do
                if (VerifyJoinCondition($E^{CR}_{i,j}$)
                &&! RedundantQuery($E^{CR}_{i,j}$)) then
                    $Q^\star \leftarrow$ ConstructQuery($E^{CR}_{i,j}$);
        $\mathcal{Q}^{rec} \leftarrow$ QueryReformulation($Q^\star$, $S^\star_j$, XMAP($S^\star_j$));
        if ($|\mathcal{Q}^{rec}| > 0$) then
            $\mathcal{Q}^\star \leftarrow \mathcal{Q}^\star \cup \mathcal{Q}^{rec}$;
        $\mathcal{Q}^\star \leftarrow \mathcal{Q}^\star \cup Q^\star$;
    return $\mathcal{Q}^\star$
end
```

Fig. 1. Pseudo-code of the XMAP reformulation algorithm. Main procedure.

C. Example

In the following we briefly describe an example of use of the XMAP algorithm.

Let suppose a user wants to find the *title* of the paper published in the *year* 2000. To this aim the following query Q is formulated over the schema UW:

Q=/uw/area/pubs/paper[year = "2000"]/title

In the first step the algorithm identifies the paths in the query:

- P_1=`/uw/area/pubs/paper/title`
- P_2=`/uw/area/pubs/paper/year`

and produces as output the set $\mathcal{P} = P_1 \cup P_2$. Next, exploiting the mappings associated to the schema UW (see Figure 2), the algorithm finds two mapping rules connecting UW to the schema $DBLP$ through the paths P_1 and P_2. More precisely, one of these rules relates P_1 to two paths in $DBLP$, respectively $P^\diamond_{1,1}$=`/dblp/article/title` and $P^\diamond_{1,2}$=`/dblp/proceedings/title`. Similarly, the other mapping rule relates P_2 to the path $P^\diamond_{2,1}$=`/dblp/article/year` and the path $P^\diamond_{2,2}$=`/dblp/proceedings/year`. So, the second step of the algorithm produces as output a candidate set composed of the paths $P^\diamond_1$ and $P^\diamond_2$ and the (candidate) schema $DBLP^\diamond$. In the considered example as the schema $DBLP^\diamond$ has correspondences for both paths P_1 and P_2, it is identified as a destination schema (step 3), so it can be used to reformulate the query Q. In particular, the algorithm (step 4), produces two direct reformulations of the query Q over the schema $DBLP$, respectively Q_{R_1} and Q_{R_2}.

Q_{R_1}=`/dblp/article[year="2000"]/title`
Q_{R_2}=`/dblp/proceedings[year="2000"]/title`

Since there are no more mapping rules involving the paths in the query Q, no further direct reformulations are produced. Then the algorithm is recursively invoked over the direct reformulations Q_{R_1} and Q_{R_2}, exploiting the mappings associated to the schema $DBLP$. Figure 1 shows the pseudo-code of the XMAP reformulation algorithm.

The problem of query answering in a semantic network is coNP-hard in the size of the data [2]. The main source of complexity is cycles in the semantic network. We avoid cycles by assuring that a specific path combination can be used only once from any path from the root of the reformulation tree to a leaf. With this termination condition the algorithm is guaranteed to find all the answers to a query when possible and obtain some answers in other cases as well.

III. XMAP Evaluation

Our aim is to observe how our approach behaves in a network of XML database systems having heterogeneous schemas. The experiments are conducted on the basis of some parameters like the average *rank* of the semantic network, the *number of nodes* and the *input query*. The average rank of the network is the average number of semantic neighbours per node. The number of nodes represents the number of different database schemas considered. We used the DBResearch data set to validate the XMAP framework based on data available on web sites concerning research in the database field (such as DBLP, ACM, etc). On the basis of these schemas, we defined XMAP mappings among subsets of schemas. We sampled this collection of mappings to experiment with different experimental configurations. The experimental results were obtained

```
<sourceSchema>uw</sourceSchema>
<Rule cardinality="Mapping1-N">
 <destinationSchema>dblp</destinationSchema>
 <sourcePath>/uw/area/pubs/paper/title</sourcePath>
 <destinationPath>/dblp/article/title</destinationPath>
 <destinationPath>/dblp/proceedings/title</destinationPath>
</Rule>
<Rule cardinality="Mapping1-N">
 <destinationSchema>dblp</destinationSchema>
 <sourcePath>/uw/area/pubs/paper/year</sourcePath>
 <destinationPath>/dblp/article/year</destinationPath>
 <destinationPath>/dblp/proceedings/year</destinationPath>
</Rule>
```

Fig. 2. Fragment of the *UW* XMAP document.

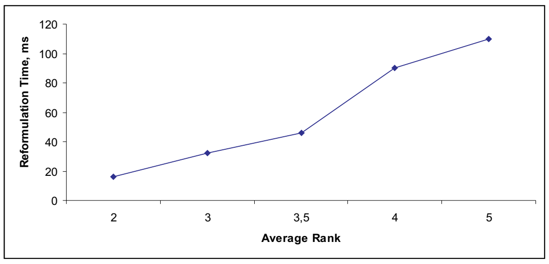

Fig. 3. Reformulation time as function of the average rank in the network.

by averaging the output of 1000 runs of a given configuration. Due to lacks of space, we show only the most significant results obtained.

Average Rank of the network. In these experiments we observed how the average rank of the semantic network affects the performance of the algorithm. To this aim we measured the reformulation time of a single query over 50 heterogeneous schemas in five configurations with different values of the average rank. Figure 3 shows that the reformulation time is a linear function of the average rank of the network and, moreover, it shows that our algorithm executes very quickly, the reformulations are obtained in at most 100 milliseconds. The raise of the shape in Figure 3 is not uniform due to the different connectivity of the mappings. A higher rank not always corresponds to a higher number of produced reformulations. This depends strongly on the mapping connectivity. Not only could some mappings provide better connectivity than others, but adding new mappings might only introduce redundant paths without contributing any new reformulation. For this reason, we introduced the relative number TT/TRQ where the total running time of the algorithm is shown as a ratio of the total number of reformulations produced. The lower this value, the higher the efficiency of the algorithm. Thus, the ratio TT/TRQ depends on the connectivity of the introduced mappings with respect to the query to reformulate as shown in Figure 4. Here, in the *uniform* set most of the mappings concern the same semantic relationships among the same concepts thus they introduce redundant paths in the semantic network and consequently they contribute few reformulations. Whereas in the *not uniform* set the percentage of redundant mappings is lower resulting in a bigger number of reformulations.

Number of Nodes. Here we aim to evaluate how the number

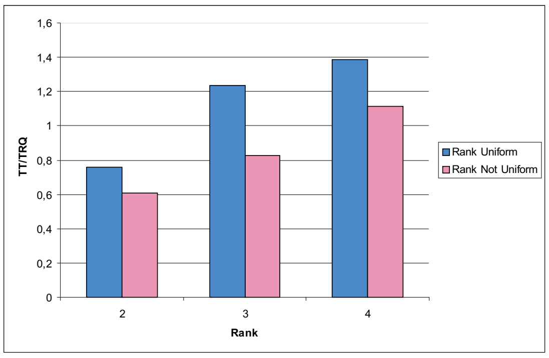

Fig. 4. Ratio TT/TRQ as function of the average rank in the network for two different mapping distributions.

of nodes in the network affects the behavior of the algorithm. We executed a query in nine configurations characterized by different values of the average rank and different number of nodes. As expected, the reformulation time grows linearly with the number of nodes in the network. However, a growing number of nodes does not affect the performance of the algorithm as it is confirmed by the trend of the ratio TT/TRQ that remains almost constant with the number of nodes (see Figure 5). Obviously, considering the contemporary increase rank and nodes we observe a raise of the value TT/TRQ due to the impact of the average rank on the reformulation time. Therefore, even if the time necessary to execute the algorithm is higher, the increasing number of produced reformulations improves the performance of the reformulation process and proves the scalability of the XMAP algorithm that is specifically tailored for networks with large number of nodes and low value of the average rank.

XMAP allows schema mappings without requiring heavyweight view definitions: although simple, the mappings are sufficient to express complex associations between XML DTDs. In XMAP, the user has only to establish simple correspondences among paths in different schemas and the system supplies the rest. Therefore, even users unfamiliar with the complex structure of the schema can provide such correspondences. Then the XMAP algorithm automatically determine rewritings of the user query, from such correspondences. From the experiments we can realize that XMAP addresses the scalability concern guaranteeing quick production of reformulations, within few milliseconds even for the most demanding configurations.

IV. A Case Study: XMAP in service-based Grids

In this section we show how a distributed and dynamic setting as the Grid could benefit from XMAP mappings. As the Grid aims at realizing the sharing and cooperation of resources among virtual organizations, when queries are posed using a node schema, answers should come from anywhere in the system. Therefore, in such a context reconciliation of schema heterogeneity plays a key role. Motivated by this issue, we developed the *Grid Data Integration System* (GDIS), a decentralized service-based data integration architecture for Grid databases; it has been presented in a previous work [6].

The GDIS system offers a wrapper/mediator-based approach to integrate data sources: it adopts the XMAP decentralized mediator approach to handle semantic heterogeneity over data sources, whereas syntactic heterogeneity is hidden behind OGSA-DAI [12] wrappers. The user query is handled by the reformulator engine that through the XMAP query reformulation algorithm produces zero, one or more reformulations of the original query. All the obtained reformulations (included the original query) are then processed independently by different processor engines through OGSA-DAI wrappers that access data source and produce the query result.

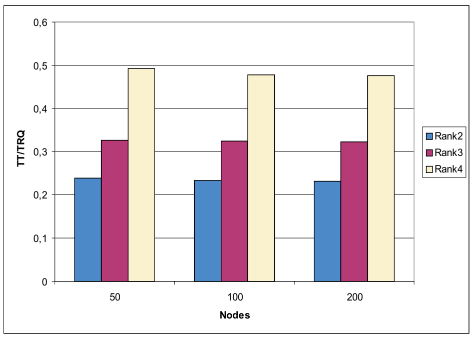

Fig. 5. Ratio TT/TRQ as function of the number of nodes for different values of the rank.

Respect to the original GDIS prototype (see [1], [6], [7], [8]), we have changed the underlying implementation as described in the following.

In previous implementations of the GDIS system, we extended the free available OGSA-DAI 5.0 Grid Data Service (GDS) reference implementation. By this, we avoided to (i) build new proprietary solutions and (ii) reimplement well solved aspects of Grid data services, and are able to concentrate on the integration task. According to this, we introduced a new activity, the *XPathQueryReformulation* activity, that wraps the XPath query reformulation algorithm of the XMAP framework. Such activity before really accesses data sources, performs schema integration by exploiting the mappings of the XML schema over which the XPath query has been formulated.

In the current version of the GDIS prototype, instead of incorporating the reformulation algorithm within the OGSA-DAI module, the XMAP algorithm is deployed as a stand-alone OGSA-DAI data Service (XMAP-ODS) that interacts with standard OGSA-DAI wrappers. Figure 6 provides an overview of the service interactions. It focuses on the interactions that concern the integration service, and thus it hides all the complexities that relate to query execution. The following architectural assumptions are made. The XMAP-ODS service has a mechanism to load local mapping information and contains a view of the schemas of the participating data resources. A database, wrapped as an OGSA-DAI resource, can join the system registering itself in a registry and informing then the XMAP-ODS service. The interactions involved when a query is issued are as follows (see also Figure 6):

1) The client contacts the *XMAP-ODS* and requests a view of the schema for each database he/she is interested in.
2) Based on the retrieved schema, he/she composes an XPath query (Q_1), which is sent to the *XMAP-ODS*.
3) The *XMAP-ODS* activates the reformulation algorithm over Q_1 (obtaining a set of reformulations of the query) and identifies the relevant sites to execute the query Q_1 by contacting the local databases via *OGSA-DAI* wrappers.
4) Each produced reformulated query (QR_i) is processed to collect results from other databases than the one initially considered by the user. To this aim the *XMAP-ODS* contacts the relevant databases through the OGSA-DAI wrappers. Query execution results are then send to the *XMAP-ODS* service that will forward them to the client.

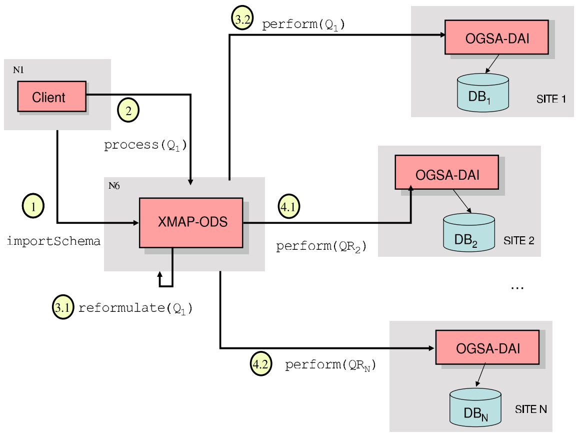

Fig. 6. Service interactions in the GDIS system

In the following we briefly present a performance evaluation of the GDIS prototype. In these experiments, we focus on the performance of GDIS as a whole, not on the specificity of the reformulation process which has been detailed in the previous section. Particularly, we will focus on the overhead incurred by extending OGSA-DAI with XMAP also measuring the cutoff of the time spent in the system.

To experiment with GDIS we have setup the following environment: the XMAP-ODS server was a Fedore Core 5 machine Pentium 4 at 3.20 GHz; the client was a Windows machine Pentium 3 at 1 GHz; the registry was a Fedore Core 4 machine Celeron at 2GHz; one OGSA-DAI server was co-located with the registry and another one was on a Fedore Core 4 machine Pentium 4 at 2.4 GHz.

The results were produced according to the following protocol: the client creates a XMAP-ODS instance, then uses it to submit the query, and finally destroys the instance it has created. More precisely, in the case considered in Figure 7, the client creates the XMAP-ODS instance, then after some time he submits the same query request 100 times (it waits for the results of the previous request before sending the next one). During the reformulation process, the XMAP-ODS service asks the registry for any mapping information it might need. It is visible on the Figure 7 that mapping information queries are performed lazily, only when it is first needed, which explains why we don't see a bunch of requests to the registry and then some quiet time to perform reformulation. Instead, reformulation CPU time is spread in between mapping information requests: as soon as a new reformulated query is found, the XMAP-ODS service asks the registry for the XMAP document of the schema over which the query is expressed (assuming it has not seen it before).

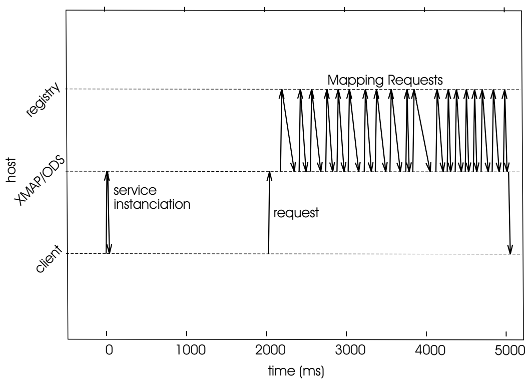

Fig. 7. Entity interactions during the reformulation of a specific query. Interactions are HTTP requests and responses.

In our experiments we considered 4 queries formulated over different database schemas. These queries were characterized by different number of produced reformulations (respectively 2, 4, 8 and 16) and the average number of the produced tuples ranged from 10 to 100. We measured the contribution to the total processing time of query reformulation, network delay and query execution. Particularly, the reformulation time is the response time to call the XMAP-ODS service and retrieve for a given XPath query the set of its equivalent queries, while the network delay also includes the time spent on the network asking and waiting for mapping documents and the execution time is the query engine time to local query execution. Figure 8 shows the minimal overhead introduced by the reformulation algorithm all along the query answering process. There it can be noted that the query execution time is the dominant cost for all the considered queries. Moreover, one should also note that the contribution to the overall query processing time of the different time components depends on the number of mappings involved in the query reformulation, the number of reformulations obtained, and on the number of database entries concerning the query answering. Local query answering time taken by the query execution engine was the bottleneck for many queries mostly for those involving a large amount of tuples.

Therefore, we can conclude that GDIS has an added-value compared to OGSA-DAI. In fact, although OGSA-DAI provides data access transparently to the user, its applicability is restricted because users typically do not know enough information about the semantics of the data in the third-party resources to which they are provided access. With our

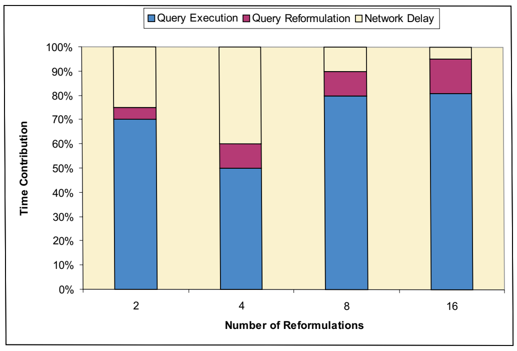

Fig. 8. Time Distribution in the GDIS system.

approach the user instead of writing a query across multiple databases has only to compose a query that refers to a single database, and the system, through the XMAP algorithm returns equivalent queries that refer to data stored to other databases and execute them automatically. Note that we do not pay a price in terms of performance because, as it is shown in Figure 8, the query reformulation time is negligible compared to the OGSA-DAI query execution time. Moreover, the importance of GDIS, is that, to the best of our knowledge, the only two works designed to provide schema-integration in Grids are the *Grid Data Mediation Service* (GDMS) that is part of the GridMiner project [13] and the SASF project [14]. Both the projects provide semantic mapping across relational databases coupled with a global-as-view approach. The main difference from GDIS is that both the approaches rely on the existence of a global schema, which is not realistic in Grids.

V. Conclusion

We have presented and evaluated the XMAP framework for P2P Schema-mapping over XML data, focusing on the following contributions: (i) a simple and intuitive P2P schema-mapping approach based on path correspondences; (ii) a query reformulation algorithm for XPath queries that is scalable and efficient, as revealed by the detailed experimentation; (iii) the incorporation of the XMAP framework within the GDIS architecture addressing schema heterogeneity among XML data sources over Grid nodes.

References

[1] C. Comito and D. Talia, "XML data integration in OGSA grids," in *Proceedings of the First VLDB Workshop on Data Management in Grids (DMG'O5)*, Sept. 2005, pp. 4–15.

[2] A. Y. Halevy, D. Suciu, I. Tatarinov, and Z. G. Ives, "Schema mediation in peer data management systems," in *ICDE*, Mar. 2003, pp. 505–516.

[3] I. F. Cruz, H. Xiao, and F. Hsu, "Peer-to-Peer Semantic Integration of XML and RDF Data Sources," in *AP2PC 2004*, July 2004.

[4] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch, "EDUTELLA: a P2P networking infrastructure based on RDF," in *WWW2002*, May 2002, pp. 604–615.

[5] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. J. Miller, and J. Mylopoulos, "The hyperion project: from data integration to data coordination." in *SIGMOD Record*, vol. 32, no. 3, 2003, pp. 53–58.

[6] C. Comito and D. Talia, "GDIS: A service-based architecture for data integration on grids," in *GADA*, Oct. 2004, pp. 88–98.

[7] ——, “Grid data integration based on schema mapping,” in *Applied Parallel Computing. State of the Art in Scientific Computing, 8th International Workshop, PARA 2006*, ser. Lecture Notes in Computer Science, vol. 4699. Springer, 2006, pp. 319–328.

[8] ——, “Data integration based on schema-mapping in service-based grids,” in *High Performance Computing and Grids in Action*, ser. Advances in Parallel Computing, L. Grandinetti, Ed., vol. 16. IOS Press, 2008, pp. 308–328.

[9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “The physiology of the grid,” Global Grid Forum, Jan. 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[10] J. Clark and S. DeRose, “XML path language (XPath) version 1.0,” W3C Recommendation, Nov. 1999, http://www.w3.org/TR/xpath.

[11] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu, “Data management for peer-to-peer computing : A vision,” in *WebDB*, June 2002, pp. 89–94.

[12] M. Antonioletti and et al., “OGSA-DAI: Two years on,” in *Global Grid Forum 10 — Data Area Workshop*, Mar. 2004, http://www.ogsadai.org.uk/.

[13] “GridMiner, http://www.gridminer.org/.”

[14] “SASF: service-based approach to schema federation, http://sasf.grid.leena34.com/.”