Data integration and query reformalution in service-based Grids: Architecture and Roadmap.

Carmela Comito, Domenico Talia
{ccomito, talia}@deis.unical.it
UNICAL
DEIS - University of Calabria
via P. Bucci 41 c, 87036, Rende, Italy

Anastasios Gounaris, Rizos Sakellariou
{gounaris, rizos}@cs.man.ac.uk
UoM
Department of Computer Science - University of Manchester
Oxford Road, Manchester M13 9PL, UK

CoreGRID Technical Report
Number TR-0013
September 6, 2005

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Data integration and query reformalution in service-based Grids: Architecture and Roadmap.

Carmela Comito, Domenico Talia
{ccomito, talia}@deis.unical.it
UNICAL
DEIS - University of Calabria
via P. Bucci 41 c, 87036, Rende, Italy

Anastasios Gounaris, Rizos Sakellariou
{gounaris, rizos}@cs.man.ac.uk
UoM
Department of Computer Science - University of Manchester
Oxford Road, Manchester M13 9PL, UK

CoreGRID TR-0013
September 6, 2005

Abstract

This report firstly summarises the work thus far on the XMAP data integration algorithm and on middleware with regard to Grid query processing services, secondly, proposes an architecture for data integration-enabled query processing on the Grid, and finally, presents a roadmap for its implementation with a view to producing an extended set of e-Services. These services will allow users to submit queries over a single database and receive the results from multiple databases that are semantically correlated with the former one.

1 Introduction

The Grid offers new opportunities and raises new challenges in data management that arise from the large scale, dynamic, autonomous, and distributed nature of data sources. A Grid can include related data resources maintained in different syntaxes, managed by different software systems, and accessible through different protocols and interfaces. Due to this diversity in data resources, one of the most demanding issue in managing data on Grids is reconciliation of data heterogeneity[13]. Therefore, in order to provide facilities for addressing requests over multiple heterogeneous data sources, it is necessary to provide data integration models and mechanisms.

Data integration is the flexible and managed federation, analysis, and processing of data from different distributed sources. In particular, the rise in availability of web-based data sources has led new challenges in data integration systems for obtaining decentralized, wide-scale sharing of data, preserving semantics. These new needs in data integration systems are also felt in Grid settings. In a Grid, a centralized structure for coordinating all the nodes may not be efficient because it can become a bottleneck and, most of all, it does not benefit from the dynamic and distributed nature of Grid resources.

The Grid community is devoting great attention toward the management of structured and semi-structured data such as relational and XML data. Two significant examples of such efforts are the *OGSA Data Access and Integration* (OGSA-DAI)[4] and the *OGSA Distributed Query Processor* (OGSA-DQP)[3] projects. However, till today only few

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

projects (e.g., [9, 7]) actually meet schema-integration issues necessary for establishing semantic connections among heterogeneous data sources.

For these reasons, we propose the use of the *XMAP* framework for integrating heterogeneous data sources distributed over a Grid. By means of this framework, we aim at developing a decentralized network of semantically related schemas that enables the formulation of distributed queries over heterogeneous data sources. We designed a method to combine and query XML documents through a decentralized point-to-point mediation process among the different data sources based on schema mappings. We offer a decentralized service-based architecture that exposes this XML integration formalism as an e-Service. We refer to this architecture as the *Grid Data Integration System* (GDIS). The infrastructure proposed exploits the middleware provided by OGSA-DQP and OGSA-DAI, building on top of them schema-integration services.

1.1 Background

The goal of a data integration system is to combine heterogeneous data residing at different sites by providing a unified view of this data. The two main approaches to data integration are federated database management systems (FDBMSs) and traditional mediator/wrapper-based integration systems.

A federated database management system (FDBMS)[21] is a collection of cooperating but autonomous component database systems (DBSs). The DBMS of a component DBS, or component DBMS, can be a centralized or distributed DBMS or another FDBMS. The component DBMSs can differ in different aspects such as data models, query languages, and transaction management capabilities.

Traditional data integration systems[19] are characterized by an architecture based on one or more mediated schemas and a set of sources. The sources contain the real data, while every mediated schema provides a reconciled, integrated, and virtual view of the underlying sources. Moreover, the system includes a set of source descriptions that provide semantic mappings between the relations in the source schemas and the relations in the mediated schemas[20].

Data integration on Grids presents a twofold characterization:

1. data integration is a key issue for exploiting the availability of large, heterogeneous, distributed and highly dynamic data volumes on Grids;

2. integration formalisms can benefit from an OGSA-based Grid infrastructure, since it facilitates dynamic discovery, allocation, access, and use of both data sources and computational resources, as required to support computationally demanding database operations such as query reformulation, compilation and evaluation.

Data integration on Grids has to deal with unpredictable, highly dynamic data volumes provided by unpredictable membership of nodes that happen to be participating at any given time. So, traditional approaches to data integration, such as FDBMS [21] and the use of mediator/wrapper middleware[20], are not suitable in Grid settings.

The federation approach is a rather rigid configuration where resources allocation is static and optimization cannot take advantage of evolving circumstances in the execution environment. The design of mediator/wrapper integration systems must be done globally and the coordination of mediators has been done by a central administrator which is an obstacle to the exploitation of evolving characteristics of dynamic environments. As a consequence, data sources cannot change often and significantly, otherwise they may violate the mappings to the mediated schema.

The rise in availability of web-based data sources has led to new challenges in data integration systems in order to obtain decentralized, wide-scale sharing of semantically-related data. Recently, several works on data management in peer-to-peer (P2P) systems are moving along this direction[5, 8, 15, 16, 17]. All these systems focus on an integration approach not based on a global schema: each peer represents an autonomous information system, and data integration is achieved by establishing mappings among the various peers.

To the best of our knowledge, there are only few works designed to provide schema-integration in Grids. The most notable ones are *Hyper*[9] and *GDMS*[7]. Both systems are based on the same approach that we have used ourselves: building data integration services by extending the reference implementation of OGSA-DAI. The *Grid Data Mediation Service* (GDMS) uses a wrapper/mediator approach based on a global schema. GDMS presents heterogeneous, distributed data sources as one logical virtual data source in the form of an OGSA-DAI service. This work is essentially different from ours as it uses a global schema. For its part, *Hyper* is a framework that integrates relational data in P2P systems built on Grid infrastructures. As in other P2P integration systems, the integration is achieved without using any hierarchical structure for establishing mappings among the autonomous peers. In that framework, the authors use a simple relational language for expressing both the schemas and the mappings. By comparison, our integration model follows as Hyper an approach not based on a hierarchical structure, however differently form Hyper it focuses on XML data sources and is based on schema-mappings that associate paths in different schemas.

The remainder of the paper is organized as follows. Section 2 presents the XMAP integration framework; the underlying integration model and the XMAP query reformulation algorithm are described. Section 3 illustrates the deployment of the XMAP framework on a service-based Grid architecture. The OGSA-DAP and OGSA-DAI existing query processing services are outlined in Section 4. Section 5 presents a simple example of applying the XMAP algorithm to OGSA-DQP supported relational databases, whereas Section 6 deals with the implementation details and roadmap and discusses how this report serves the purpose of the CoreGrid project and relates to other project activities.

2 A Decentralized XML Data Integration Framework

In this section, we describe a framework meant to integrate heterogeneous XML data sources distributed among nodes of a Grid. The primary design goal of this framework is to develop a decentralized network of semantically related schemas that enables the formulation of queries over heterogeneous, distributed data sources.

2.1 Integration Model

Our integration model is based on schema mappings to translate queries between different schemas. The goal of a schema mapping is to capture structural as well as terminological correspondences between schemas.

As mentioned before, traditional centralized architecture of data integration systems is not suitable for highly dynamic and distributed environments such as the Grid. Thus, we propose a decentralized approach inspired from[16] where the mapping rules are established directly among source schemas without relying on a central mediator or a hierarchy of mediators. In consequence, in our integration 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). To integrate a source in the system, one needs only to provide a set of mapping rules that describes the relationships between its schema and the other schemas it is related to.

The specification of mappings is thus flexible and scalable: each source schema is directly connected to only a small number of other schemas. However, it remains reachable from all other schemas that belong to its transitive closure. 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”.

We address structural heterogeneity among XML data sources by associating paths in different schemas. Mappings are specified 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 is the element denoting 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 structure of XMAP documents is conform to the schema shown in Figure 1. One can notice the presence of a single `sourceSchema` element, and a set of `Rule` elements defining the mapping rules. `Rule` elements have a complex structure which specifies the paths involved in the mappings and the cardinality constraints among them.

```
<schema targetNamespace="http://XMAP/XMAPDocument"
        xmlns="http://www.w3.org/2001/XMLSchema" ?>
 <element name="Mapping">
  <complexType>
   <sequence>
    <element name="sourceSchema" type="string"
             minOccurs="1" maxOccurs="1"/>
    <element name="Rule" minOccurs="1">
     <complexType>
      <sequence>
       <attribute name="Cardinality" type="string"
                  minOccurs="1" maxOccurs="1"/>
       <element name="sourcePath" type="string" minOccurs="1"/>
       <element name="destSchema" type="string"
                minOccurs="1" maxOccurs="1"/>
       <element name="destPath" type="string" minOccurs="1"/>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </complexType>
 </element>
</schema>
```

Figure 1: XML schema for XMAP documents.

2.2 A Reformulation Algorithm for XPath Queries

In this section we present an algorithm to reformulate an XPath query on the basis of the mapping rules established for the schema over which the query is formulated. In the following, we suppose that we have a set of XML data sources, that each data source is compliant to an XML Schema and that, for each schema, an XMAP document containing the mappings related to this schema is provided.

Our query processing approach exploits the semantic connections established in the system by performing the *query reformulation algorithm* before executing the 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. Thus, the user can retrieve data from all the related sources in the system by simply submitting a single XPath query.

The query reformulation algorithm uses as input an XPath query and the mappings, and it produces as output zero, one or more reformulated queries. We describe now in details the logic of the algorithm. In this discussion, we use Q to denote the input XPath query, S the source schema over which Q is formulated, M the mappings in the system and Q_{R_i} the reformulated queries produced by the algorithm.

The algorithm can be decomposed in the following stages:

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 candidate 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. More precisely, for each P_i of the query Q, the algorithm looks for all corresponding paths in the schemas transitively connected to S through path P_i. These paths $P^{\diamond}_{i,j}$ are called *candidate paths*, and the schema $S^{\diamond}_j$ they belong to, *candidate schema*. In particular we define a *candidate 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 $S^{\diamond}_j$. So, for each path expression $P_i \in \mathcal{P}$, zero, one or more candidate elements $E^{\diamond}_{i,j}$ are built with $0 \leq j \leq n$ (n is the number of source schemas in the system). A *candidate set* $\mathcal{E}^{\diamond}$ is a set of candidate elements $\{E^{\diamond}_1, \dots E^{\diamond}_n\}$ (with $E^{\diamond}_j = \bigcup_i E^{\diamond}_{i,j}$). Thus, this stage returns as a result the set $\mathcal{E}^{\diamond}$.

3. *Pruning of candidate schemas.*
   The third stage of the algorithm checks for each candidate 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 candidate schema has at least one candidate path for each path present in the query. Moreover, it needs to make sure that none of these candidate paths has already been used to rewrite Q in order to avoid considering redundant paths. The schemas that meet both these conditions are the only ones that we will be considered to obtain reformulated queries, we call them destination schemas. We define a *destination 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 $S^{\star}_j$. So, for each path expression P_i in $\mathcal{P}$, zero, one or more candidate elements $E^{\star}_{i,j}$ are built where $0 \leq j \leq |\mathcal{E}^{\diamond}|$. A *destination set* $\mathcal{E}^{\star}$ is a set of destination elements $\{E^{\star}_1, \dots E^{\star}_n\}$ (with $E^{\star}_j = \bigcup_i E^{\star}_{i,j}$). Thus, this stage returns as a result the set $\mathcal{E}^{\star}$.

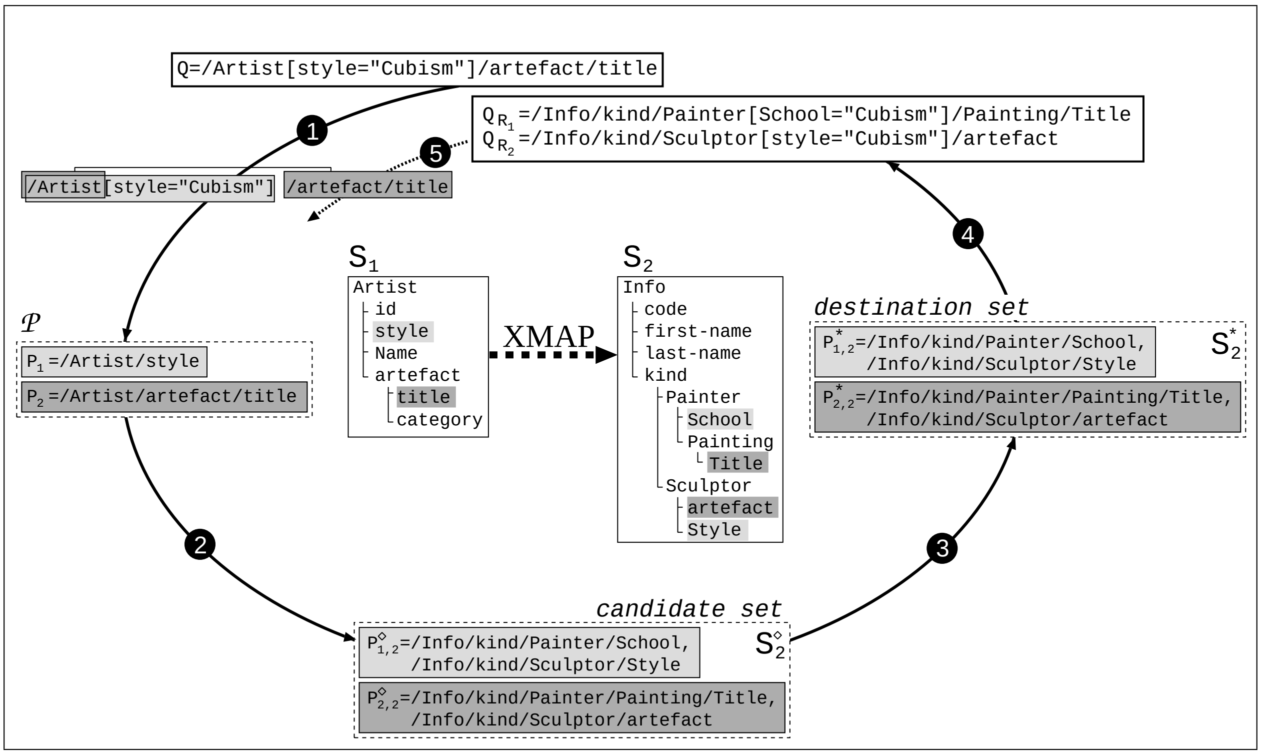
4. *Constructing reformulated queries.*
   In this stage, given the set $\mathcal{E}^{\star}$, the algorithm produces one or more XPath queries over each schema in the set. More precisely, for each destination schema $S^{\star}_j$ in $\mathcal{E}^{\star}$ the following steps are performed:

   - *Assessing cardinality constraints.* If each path P_i in $\mathcal{P}$ has a single correspondent path $P^{\star}_{i,j}$ the schema $S^{\star}_j$, all the mapping rules between S and $S^{\star}_j$ will be of kind 1-1 or N-1. Thus, the output of the reformulation will be a single query expressed over the destination schema $S^{\star}_j$. At the opposite, if there exists (at least) one path in $\mathcal{P}$ that has more than one destination path over the schema $S^{\star}_j$ (1-N mapping), there will be more than one reformulated query over the schema $S^{\star}_j$. The number of reformulated queries depends on the possible path combinations. If k is the cardinality of the set $\mathcal{P}$, $|\mathcal{P}|$, the number com of possible distinct path combinations for $S^{\star}_j$, is equal to the product of the cardinality of each path $P^{\star}_{i,j}$, $com = |P^{\star}_{1,j}| \times |P^{\star}_{2,j}| \times \cdots \times |P^{\star}_{k,j}|$.
   - *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 destination schema $S^{\star}_j$. In the 1-1 and N-1 mapping cases, since there will be a single reformulated query, it will not be possible to reformulate the query with respect to the schema $S^{\star}_j$ if there exist at least two paths for which join conditions are not respected. Finally, in the 1-N mapping case, there will be as many reformulated queries as there exist satisfied join conditions among the paths of each combination.
   - *Composing XPath Queries.* Once the join conditions between the destination paths have been checked, 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 $S^{\star}_j$.

5. *Recursive invocation of the algorithm.*
   The algorithm is recursively invoked over the reformulated queries in order to produce the queries corresponding to every transitive mappings.

In Figure 2 is described an example of use of the XMAP algorithm. Here a query Q is formulated over the schema S_1. In the first step the algorithm identifies the paths P_1 and P_2 in Q and produces as output the set $\mathcal{P}$. Next, exploiting the XMAP document associated to the schema S_1, the algorithm finds two mapping rules connecting S_1 to S_2 trough the paths P_1 and P_2. More precisely, one of these rules relates P1 to two paths in S_2, respectively `/Info/kind/Painter/School` and `/Info/Kind/Sculptor/Artefact`. Similarly, the other mapping rules relates P_2 to `/Info/Kind/Painter/Painting/Title` and `/Info/Kind/Sculptor/artefact`. So, the second step of the algorithm produces as output a candidate set composed of the elements $P^{\diamond}_{i,j}$ and the (candidate) schema $S^{\diamond}_2$. In the considered example as the schema $S^{\diamond}_2$ 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 produces two reformulations of the query Q over the schema S_2, respectively Q_{R_1} and Q_{R_2}.

In Figure 3 is shown the pseudo-code of the XMAP reformulation algorithm.



Q=/Artist[style="Cubism"]/artefact/title
Q_R1=/Info/kind/Painter[School="Cubism"]/Painting/Title
Q_R2=/Info/kind/Sculptor[style="Cubism"]/artefact
1
2
3
4
5
/Artist[style="Cubism"]
/artefact/title
S1
S2
Artist
id
style
Name
artefact
title
category
XMAP
Info
code
first-name
last-name
kind
Painter
School
Painting
Title
Sculptor
artefact
Style
𝒫
P1=/Artist/style
P2=/Artist/artefact/title
destination set
P*1,2=/Info/kind/Painter/School, /Info/kind/Sculptor/Style
P*2,2=/Info/kind/Painter/Painting/Title, /Info/kind/Sculptor/artefact
S*2
candidate set
P◇1,2=/Info/kind/Painter/School, /Info/kind/Sculptor/Style
P◇2,2=/Info/kind/Painter/Painting/Title, /Info/kind/Sculptor/artefact
S◇2

Figure 2: Example of use of the XMAP algorithm.

3 GDIS: A high-level architecture for data integration in Grids

In this section, we describe a decentralized service-based data integration architecture for Grid databases that we refer to as *Grid Data Integration System* (GDIS)[12]. The main concern of such system is reconciliation of data source heterogeneity among distributed databases.

3.1 GDIS Logical Model and Abstract Architecture

We designed the GDIS system as a set of Grid nodes that accomplish one or more of the typical tasks of a data integration system. The main entities and associated tasks in GDIS are: (1) *data providers* supplying data source, (2) *mediators* whose source schema transitively connects two sources , (3) *clients* formulating queries.

The GDIS system offers a wrapper/mediator-based approach to integrate data sources: it adopts a decentralized mediator approach to handle semantic heterogeneity over data sources, whereas syntactic heterogeneity is hidden behind ad-hoc wrappers. More precisely, the proposed framework is characterized by three core components: a

```
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
        $\mathcal{E}^\diamond \leftarrow$ FindCandidatePath($P_i$, $M$);
    $\mathcal{E}^\star \leftarrow$ PruningSchema($\mathcal{E}^\diamond$);
    for each $S_j^\star \in \mathcal{E}^\star$ do
        if (Mapping1-N(S, $S_j^\star$)) then
            $\mathcal{Q}^\diamond \leftarrow$ CombinePaths($E_{i,j}^\star$);
            for each candidate query $Q^\diamond \in \mathcal{Q}^\diamond$ do
                if (VerifyJoinCondition($Q^\diamond$)) then
                    $Q^\star \leftarrow$ ConstructQuery($Q^\diamond$);
                    $\mathcal{Q}^{rec} \leftarrow$ QueryReformulation($Q^\star$, $S_j^\star$, XMAP($S_j^\star$));
                    if ($|\mathcal{Q}^{rec}| > 0$) then
                        $\mathcal{Q}^\star \leftarrow \mathcal{Q}^\star \cup \mathcal{Q}^{rec}$;
                    $\mathcal{Q}^\star \leftarrow \mathcal{Q}^\star \cup Q^\star$;
        else
            if (VerifyJoinCondition($E_{i,j}^\star$)) then
                $Q^\star \leftarrow$ ConstructQuery($Q^\diamond$);
                $\mathcal{Q}^{rec} \leftarrow$ QueryReformulation($Q^\star$, $S_j^\star$, XMAP($S_j^\star$));
                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
```

Figure 3: Pseudo-code of the XMAP reformulation algorithm.

query reformulation engine, a *wrapper module*, and a *distributed query processor*. As a consequence of these features, some nodes in the system supply appropriate services addressing the challenges characterizing the design of a wrapper/mediator-based data integration system. Thus, we can further classify the nodes in the system on the basis of the functionality they achieve with regard to the above mentioned features, obtaining in such way the following categorization:

- *processing nodes* that supply distributed query processing capabilities aiming at compile, optimise, partition and schedule distributed query execution plans over multiple *execution nodes*; every execution node is in charge of a sub-query execution plan assigned to it by a processing node;
- *data integration nodes* that offer: (i) a set of data integration utilities allowing to establish mappings, and (ii) the query reformulation algorithm introduced by the XMAP integration formalism.
- *wrapper nodes* allowing for access actual data sources.

It should be noted that any node can carry out all the mentioned services.

Figure 1 shows the abstract architecture of the GDIS system. All the kinds of nodes in the system and typical interactions that take place among them are shown. As above mentioned, a node in the system can assume several different roles on the basis of the reachable functionality. In the described scenario a client node CN, formulates a query over the schema contributed by the mediator node MN and, then, it poses the query (interaction (1)) to the node IN holding a reformulator engine. The reformulator performs the XMAP query reformulation algorithm that accepts as input the query and the mappings relative to the schema over which the query has been formulated. (interaction (2)). After the reformulated query has been produced, the node IN invokes (interaction (3)) the distributed query processor (DQP) on the node PN. The DQP performs the so far produced reformulated query delegating and scheduling portions of the query plan over multiple execution nodes (interaction (4)). Every execution node (EN), handling a partition of the query execution plan assigned to it by the processing node, accesses actual data stored in the database. To this aim each node EN (interaction (5)) uses the wrapper made available by the node WN that will access data held in the database on node DN (interactions (6-7)) and return it (interaction (8)) to the node EN. After that, the sub-query results begin to propagate across the execution nodes (interaction (9)) until reaching the node PN that collects them into the query result and passes it to the querying node IN (interaction (10)). Finally, the node IN propagates the query answer (interaction (11)) to the client (node CN). Note that for clarity reasons in the described scenario it is assumed only one reformulation of the submitted query. Moreover, note that, the dotted arrow among the node DN and the node MN means that there are semantic mappings among the schemas exported by the two nodes.

3.2 GDIS Architecture Implementation

We designed our data integration system as a service-based distributed architecture. So, the proposed architecture is general enough to be feasibly adopted both in the two implementations of the Globus Toolkit, GT3 and GT4. At present, the system is integrated with GT3 based on OGSA; minor modifications will be required to port the system to the WSRF-based GT4. In particular, in the Globus Toolkit 4 Grid services will be replaced by WS-Resources, service data by resource properties and Index Services by ServiceGroups.

Following the GT3 terminology, each node in GDIS exposes all its resources as Grid services except data resources and data integration facility that are exposed as Grid Data Services (GDSs). In so doing, the GDIS system introduces the OGSA-based *Grid Data Integration* (GDI) service that service that extends OGSA-DAI and OGSA-DQP portTypes with additional functionality both to enable users to specify semantic mappings (in the form of XMAP documents) among a set of data sources, and to execute the XMAP query rewriting algorithm.

Every GDI instance, other than introducing specific portTypes, can define some portTypes from OGSA-DAI and OGSA-DQP. As a Grid service, GDI must define a *GridService* portType, and can define optional portTypes, such as *NotificationSink* an *NotificationSource*. Furthermore, GDI implements the portTypes *GDS* and *GDT* from OGSA-DAI, and the portType *GDQ* from OGSA-DQP. To these portTypes, it adds the following new ones:

- the *Import Mappings* (*IM*) portType, through which a client can import the mappings stored in the other GDI services in the system. The imported mappings will be exposed by the GDI as SDEs;
- the *Manual Mappings Composition* (*MMC*) portType, through which a client manually builds schema mappings stored in the form of an XMAP document as SDEs;
- the *Query Reformulation Algorithm* (*QRA*) portType, that performs the XMAP query reformulation algorithm and receives as input a query, schema mappings and produces as output a reformulated query. The reformulated query will be the input of the *GDS* portType offered by the OGSA-DQP GDQS service.

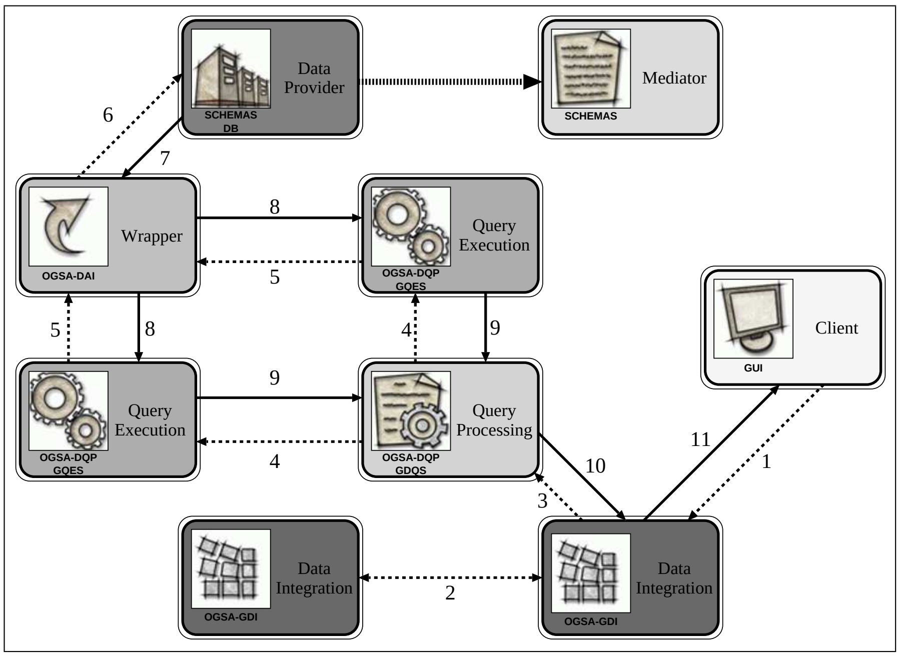


Figure 4: GDIS functional architecture.

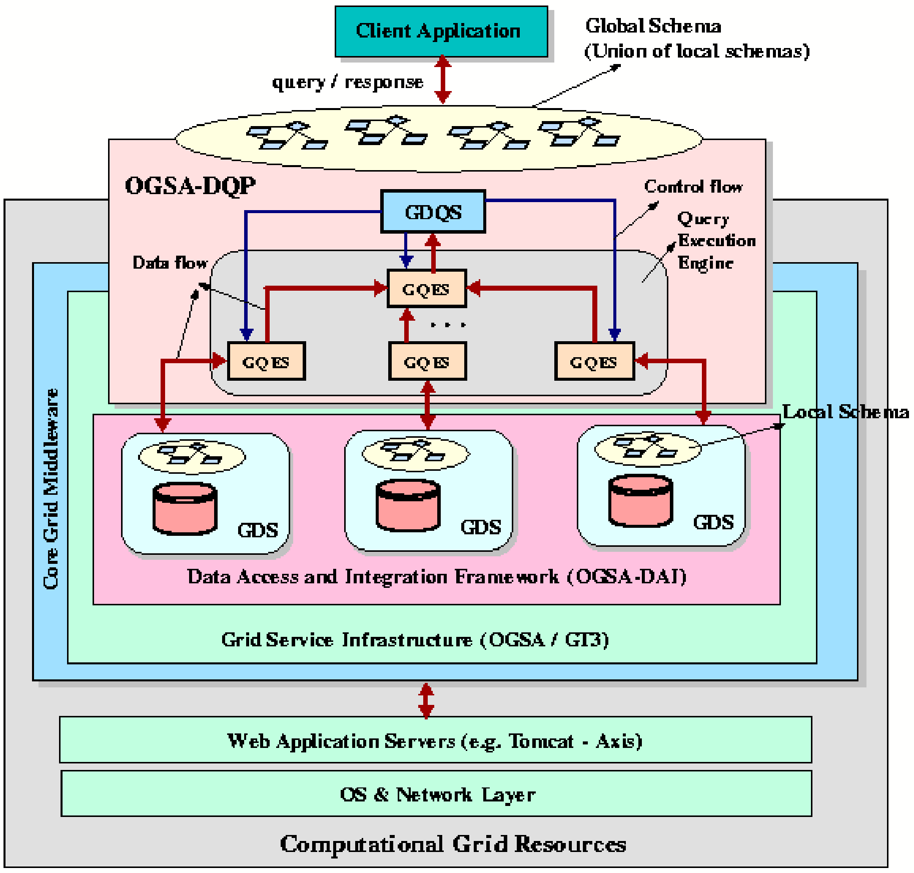


Figure 5: The architecture of OGSA-DQP.

4 Introduction to Grid query processing services

OGSA-DQP is an open source service-based Distributed Query Processor; as such, it supports the evaluation of queries over collections of potentially remote data access and analysis services. OGSA-DQP uses Grid Data Services (GDSs) provided by OGSA-DAI to hide data source heterogeneities and ensure consistent access to data and metadata. The current version of OGSA-DQP, OGSA-DQP 2.0, uses Globus Toolkit 3.2 for grid service creation and management. Thus OGSA-DQP builds upon an OGSA-DAI distribution that is based on the OGSI infrastructure. In addition, both GT3.2 and OGSA-DAI require a web service container (e.g. Axis) and a web server (such as Apache Tomcat) below them (see Figure 5). A forthcoming release of OGSA-DQP, due in fall of 2005, will support the WS-I and WSRF platforms as well.

OGSA-DQP provides two additional types of services, Grid Distributed Query Services (GDQSs) and Grid Query Evaluation Services (GQESs). The former are visible to end users through a GUI client, accept queries from them, construct and optimise the corresponding query plans and coordinate the query execution. GQESs implement the query engine, interact with other services (such as GDSs, ordinary Web Services and other instances of GQESs), and are responsible for the execution of the query plans created by GDQSs.

5 Integrating the XMAP algorithm in service-based Grids: A walk-through example

The XMAP algorithm can be used for data integration-enable query processing in OGSA-DQP

This example aims to show how the XMAP algorithm can be applied on top of the OGSA-DAI and OGSA-DQP

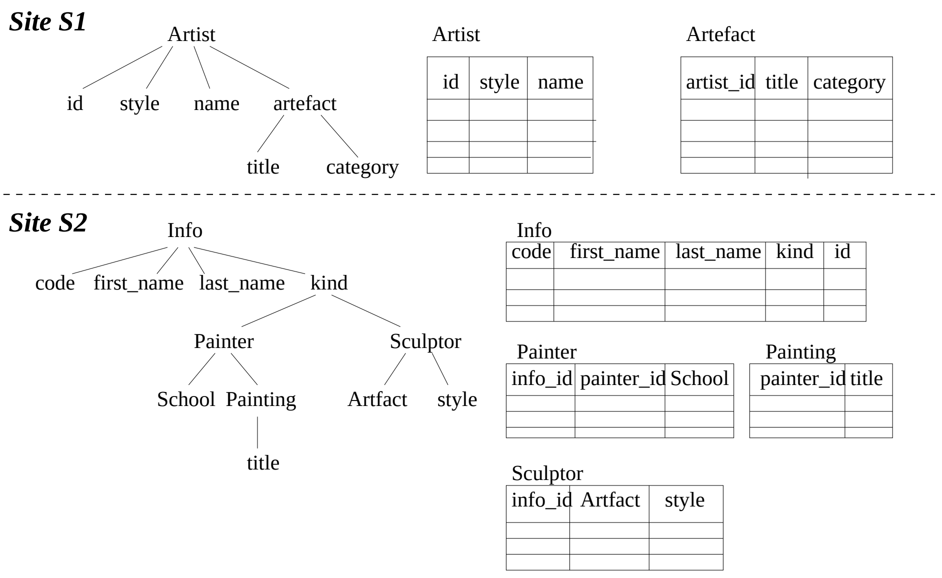


Figure 6: The example schemas.

services.

5.1 The database schemas

In the example, we will assume that the underlying databases, of which the XML representation of the schema is processed by the XMAP algorithm, are, in fact, relational databases, like those supported by the current version of OGSA-DQP.

We assume that there are two sites, each holding a separate, autonomous database that contains information about artists and their works[1]. Figure 6 presents two self-explanatory views: one hierarchical (for native XML databases), and one tabular (for object-relational DBMSs).

In OGSA-DQP, the table schemas are retrieved and exposed in the form of XML documents, as shown in Figure 7.

5.2 The XMAP mappings and the query reformulation

The XMAP mappings need to capture the semantic relationships between the data fields in different databases, including the primary and foreign keys. This can be done in two ways, which are illustrated in Figures 8 and 9, respectively. Both the ways seem to be feasible. However, the second one is slightly more comprehensible, and thus more desirable.

The actual query reformulation occurs exactly as described in [11]. Initially, the users submit XPath queries that refer to a single physical database. E.g., `/S1/Artist[style="Cubism'']/name` extracts the names of the artists whose style is Cubism and their data is stored in the *S1* database. Similarly, `/S1/Artefact/title` returns the titles of the artifacts in the same database. When the XMAP algorithm is applied for the second query, two more XPath expressions will be created that refer to the *S2* database: `/S2/Painting/Title` and `/S2/Sculptor/Artefact`. At the back-end, the following queries will be submitted to the underlying databases (in SQL-like format):

`select title from Artefact;`,
`select title from Painting;`, and
`select Artefact from Sculptor;`

Note that the mapping of simple XPath expressions to SQL/OQL is feasible [18].

[1]The example refers to another version of the example mentioned in the XMAP paper [11] and in Section 2.

```
<databaseSchema dbname="S1">
    <table name="Artist">
        <column name="id" />
        <column name="style" />
        <column name="name" />
        <primaryKey>
            <columnName>id</columnName>
        </primaryKey>
    </table>
    <table name="Artefact">
        <column name="artist_id" />
        <column name="title" />
        <column name="category" />
    </table>
</databaseSchema>

<databaseSchema dbname="S2">
    <table name="Info">
        <column name="id" />
        <column name="code" />
        <column name="first_name" />
        <column name="last_name" />
        <column name="kind" />
        <primaryKey>
            <columnName>id</columnName>
        </primaryKey>
    </table>
    <table name="Painter">
        <column name="painter_id" />
        <column name="info_id" />
        <column name="school" />
        <primaryKey>
            <columnName>painter_id</columnName>
        </primaryKey>
    </table>
    <table name="Painting">
        <column name="painter_id" />
        <column name="title" />
        <primaryKey>
            <columnName>title</columnName>
        </primaryKey>
    </table>
    <table name="Sculptor">
        <column name="info_id" />
        <column name="artefact" />
        <column name="style" />
    </table>
</databaseSchema>
```

Figure 7: The XML representation of the schemas of the example databases.

```
i)   databaseSchema[@dbname=S1]/table[@name=Artist]/column[@name=style] ->
     databaseSchema[@dbname=S2]/table[@name=Painter]/column[@name=school],
     databaseSchema[@dbname=S2]/table[@name=Sculptor]/column[@name=style]

ii)  databaseSchema[@dbname=S1]/table[@name=Artefact]/column[@name=title] ->
     databaseSchema[@dbname=S2]/table[@name=Painting]/column[@name=title],
     databaseSchema[@dbname=S2]/table[@name=Sculptor]/column[@name=artefact]

iii) databaseSchema[@dbname=S1]/table[@name=Artist/column[@name=id ->
     databaseSchema[@dbname=S2]/table[@name=Info/column[@name=id]

iv)  databaseSchema[@dbname=S1]/table[@name=Artefact]/column[@name=artist_id] ->
     databaseSchema[@dbname=S2]/table[@name=Painter]/column[@name=info_id],
     databaseSchema[@dbname=S2]/table[@name=Sculptor]/column[@name=info_id]
```

Figure 8: The XMAP mappings.

```
i)   S1/Artist/style -> S2/Painter/school, S2/Sculptor/style

ii)  S1/Artefact/title -> S2/Painting/title, S2/Sculptor/artefact

iii) S1/Artist/id -> S2/Info/id

iv)  S1/Artefact/artist_id -> S2/Painter/info_id, S2/Sculptor/info_id
```

Figure 9: A simpler form of the XMAP mappings.

6 Implementation Roadmap

6.1 Service Interactions and System Design

The XMAP query reformulation algorithm is deployed as a stand-alone service, called *Grid Data Integration service (GDI)*. Figure 10 provides an overview of the service interactions involved in the incorporation of data integration functionality in distributed query processing on the Grid. It focuses on the interactions that concern the *GDI*, and thus it hides all the complexities that relate to (distributed) query submission and execution. As such, it complements the service interactions between the OGSA-DAI and DQP services, which are described in detail in [2].

The following architectural assumptions are made. The *GDI* is deployed at each site participating in a dynamic database federation and has a mechanism to load local mapping information, based on the interface described in Section 3. Following the Globus Toolkit 3 [1] terminology, it implements additional *port-types* (see sect 3.2), among which the *Query Reformulation Algorithm(QRA)* port-type, which accepts XPath expressions, applies the XMAP algorithm to them, and returns the results. A database can join the system as in OGSA-DQP: registering itself in a registry and informing the *GDQS*. The only difference is that, given the assumptions above, it should be associated with both a *GQES* and a *GDI*.

Also, there is one *GQES* per site to evaluate (sub)queries, and at least one *GDQS*. As in classical OGSA-DQP scenarios, the *GDQS* contains a view of the schemas of the participating data resources, and a list of the computational resources that are available. The users interact only with this service from a client application that need not be exposed as a service.

The interactions are as follows (see also Figure 10):

1. The client contacts the *GDQS* 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, which is sent to the *GDQS*.
3. The *GDQS* transforms, parses, optimises, schedules and compiles a query execution plan [22]. This process entails the identification of the relevant sites, and consequently their local *GQES* and *GDI*. The resulting query execution plan is sent to the corresponding *GQES*, which returns the results asynchronously, after contacting the local database via a *GDS*.

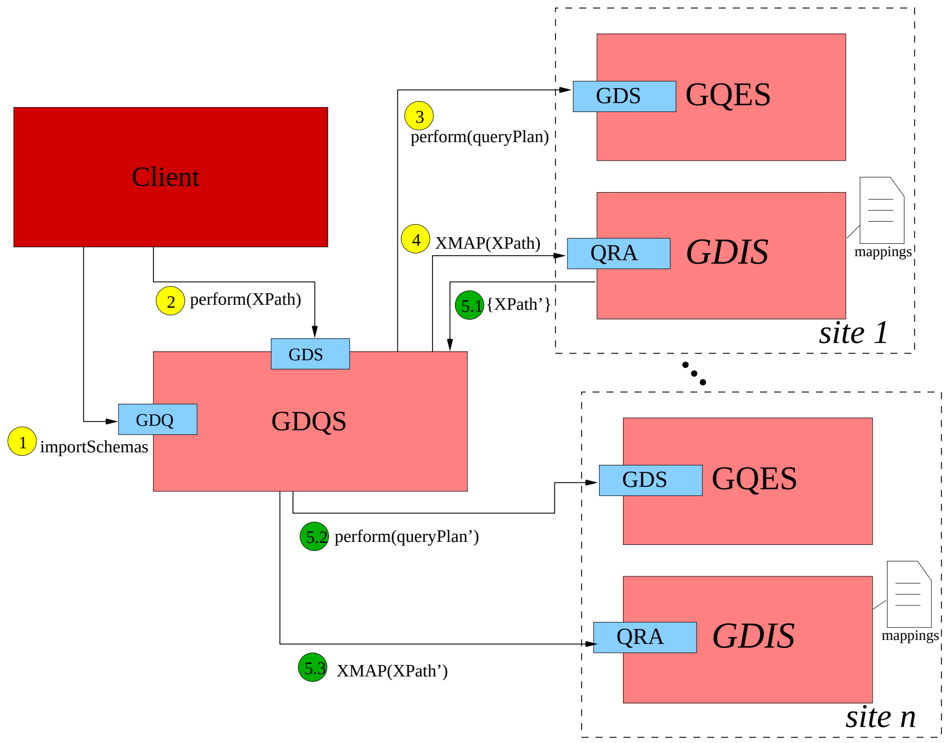


Figure 10: Data integration-enabled query processing on the Grid: service interactions.

4. The initial XPath expression is sent to the *GDI* that is co-located with the *GQES* of the previous step to perform the XMAP algorithm.

5. As long as the call to the *GDI* returns at least one XPath expression that has not been considered yet in the same session, the following steps are executed in an iterative manner.

   (a) The results of the call to the *GDI* are collected by the *GDQS*. They contain a set of XPath expressions. The *GDQS* filters out the ones that have already been processed in the current session.

   (b) Each remaining XPath expression is processed as in Step 3 to collect results from other databases than the one initially considered by the user.

   (c) The same XPath expression is processed as in Step 4 to find additional correlated queries.

6.2 A Summary of the Extensions Envisaged to the Current Querying Services

The afore-mentioned architecture, apart from the development of the new *GDI* service, implies some extensions to the current services and clients that are available from OGSA-DAI and OGSA-DQP. These extensions are, in our view, reasonable and feasible, and thus make the overall proposal of practical interest. They are summarised below:

- The client should expose the schemas per database and rather than as a unified view.
- *GDQS* should be capable of accepting XPath queries, and of transforming these XPath queries to OQL before parsing, compiling, optimising and scheduling them. Such a transformation falls in an active research area (e.g., [14, 6]), and will be realised as an additional component within the query compiler.
- *GDQS* should implement an additional XMAP-related activity that, given an XPath expression, finds the corresponding *GDI*, and calls the XMAP on it. This returns a set of corresponding XPaths.
- The client should be capable of aggregating results stemming from multiple queries.

- *GDQS* should be capable of accepting requests that contain more than one (XPath) statement.
- Also, *GDI* should be capable of processing requests that clean, update and install mapping documents.

6.3 Looking Ahead

The proposed architecture will provide added value to the existing querying services, and increase the scope of the applications that may use them. It will result in a middleware infrastructure that can be enhanced with more functionality. With a view to incorporating more features, the following stages of extensions have been identified:

Stage A: XPath is a simple language, and, as such, it cannot cover many of the common user requests. Allowing more complex user queries to be submitted, and using the same XMAP algorithm that relies on paths, is a challenging problem. It is expected that more extensive use of the knowledge about key/foreign-key relationships will be required in order to reformulate more expressive queries (such as XQuery, SQL and OQL correctly).

Stage B: OGSA-DQP naturally provides the capability to submit queries over distributed sources in a manner that is transparent to the user. In order to use this functionality in the future, some (non-extensive) changes in the validity criteria of reformulated queries in the XMAP algorithm will be required.

Stage C: A more challenging problem is to allow initial queries to be distributed. This raises a new set of issues, which include which site should hold the mappings, whether any more metadata at the GDQS-level is required, and how non-duplicate results can be guaranteed.

Stage D: Finally, we plan to explore alternative architectures, and especially architectures in which the *GDIs* are may not be co-located with *GQESs*, and can be shared between multiple sites.

6.4 Benefits for the CoreGRID activities

The massive amount of data sets that today is available from geographically distributed storage sources, is making data integration as important as data mining and knowledge discovery for exploiting the value of such large and distributed data repositories. Integration and correlation of large data sets demand significant advances in middleware for integrating data from diverse distributed sources. So, data integration is a key element for data management and knowledge discovery in Grids.

The primary objective of the CoreGRID Network of Excellence is to build solid methodological and technological foundations for GRID computing services that will remain at the forefront of excellence. In particular, the objective of the WP2 is to provide a collaborative infrastructure for European research teams working on the distributed storage management of GRIDs, the programming techniques and tools to support data intensive applications, and the integration of data and computational GRIDs with information and knowledge GRIDs.

The issues described in the report meet the main research topics included in the WP2 Roadmap. More precisely, with the research work presented in the report we address many of the objectives of both tasks 2.2 and 2.3.

Pertinent objectives of task 2.2 are organized along the following axis:

- Standardization and Integration : extending and standardizing the existing OGSA middleware for knowledge-based GRID services;
- Data GRID management: implementing and extending OGSA-DAI and OGSA-DQP for supporting data-intensive applications on GRIDs;
- Service-based data integration on GRIDs: scalable P2P models and architectures for distributed data integration.

Respectively, objectives of task 2.3 that are pertinent to our work are the following:

- Semantic Mapping: Representation of relationships between different GRID entities and resources;
- Intelligent queries: Query mechanism and intelligent agents for query formation.

References

[1] The Globus toolkit, http://www.globus.org.

[2] M. Nedim Alpdemir, A. Mukherjee, Norman W. Paton, Paul Watson, Alvaro A. A. Fernandes, Anastasios Gounaris, and Jim Smith. Service-based distributed querying on the grid. In Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors, *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, pages 467–482. Springer, 2003.

[3] M. Nedim Alpdemir, Arijit Mukherjee, Anastasios Gounaris, Norman W. Paton, Paul Watson, Alvaro A. A. Fernandes, and Desmond J. Fitzgerald. OGSA-DQP: A service for distributed querying on the grid. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology*, pages 858–861, March 2004.

[4] Mario Antonioletti and et al. OGSA-DAI: Two years on. In *Global Grid Forum 10 — Data Area Workshop*, March 2004.

[5] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB 2002)*, pages 89–94, June 2002.

[6] Kevin S. Beyer, Roberta Cochrane, Vanja Josifovski, Jim Kleewein, George Lapis, Guy M. Lohman, Bob Lyle, Fatma Ozcan, Hamid Pirahesh, Norman Seemann, Tuong C. Truong, Bert Van der Linden, Brian Vickery, and Chun Zhang. System rx: One part relational, one part xml. In *SIGMOD Conference 2005*, pages 347–358, 2005.

[7] P. Brezany, A. Woehrer, and A. M. Tjoa. Novel mediator architectures for grid information systems. *Journal for Future Generation Computer Systems - Grid Computing: Theory, Methods and Applications.*, 21(1):107–114, 2005.

[8] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in P2P systems. In *Proceedings of the First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*, pages 77–90, September 2003.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. Hyper: A framework for peer-to-peer data integration on grids. In *Proc. of the Int. Conference on Semantics of a Networked World: Semantics for Grid Databases (ICSNW 2004)*, volume 3226 of *Lecture Notes in Computer Science*, pages 144–157, 2004.

[10] James Clark and Steve DeRose. XML path language (XPath) version 1.0. W3C Recommendation, November 1999. `http://www.w3.org/TR/xpath`.

[11] C. Comito and D. Talia. Xml data integration in ogsa grids. In *1st Int. Workshop on Data Management in Grids (to appear)*, 2005.

[12] Carmela Comito and Domenico Talia. GDIS: A service-based architecture for data integration on grids. In *Proceedings of the Workshop on Grid Computing and Its Application to Data Analysis*, pages 88–98, October 2004.

[13] Karl Czajkowski and et al. The WS-resource framework version 1.0. The Globus Alliance, Draft, March 2004. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`.

[14] Wenfei Fan, Jeffrey Xu Yu, Hongjun Lu, and Jianhua Lu. Query translation from xpath to sql in the presence of recursive dtds. In *VLDB Conference 2005*, 2005.

[15] Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proceedings of the First International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*, pages 64–76, September 2003.

[16] Alon Y. Halevy, Dan Suciu, Igor Tatarinov, and Zachary G. Ives. Schema mediation in peer data management systems. In *Proceedings of the 19th International Conference on Data Engineering*, pages 505–516, March 2003.

[17] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 325–336, June 2003.

[18] George Lapis. Xml and relational storage - are they mutually exclusive? available at http://www.idealliance.org/proceedings/xtech05/papers/02-05-01/ (accessed in july 2005).

[19] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 233–246, June 2002.

[20] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96)*, pages 251–262, September 1996.

[21] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[22] Jim Smith, Anastasios Gounaris, Paul Watson, Norman W. Paton, Alvaro A. A. Fernandes, and Rizos Sakellariou. Distributed query processing on the grid. In Manish Parashar, editor, *Grid Computing - GRID 2002, Third International Workshop, Baltimore, MD, USA, November 18, 2002, Proceedings*, pages 279–290. Springer, 2002.