iXPeer: Implementing layers of abstraction in P2P Schema Mapping using AutoMed

Zohra Bellahsène
LIRMM
UMR 5506 CNRS
Univ. Montpellier II
France
bella@lirmm.fr

Charalambos Lazanitis
Dept. Computing
Imperial College London
London SW7 2AZ
UK
cl201@doc.ic.ac.uk

Peter McBrien
Dept. Computing
Imperial College London
London SW7 2AZ
UK
pjm@doc.ic.ac.uk

Nikos Rizopoulos
Dept. Computing
Imperial College London
London SW7 2AZ
UK
nr600@doc.ic.ac.uk

ABSTRACT

The task of model based data integration becomes more complicated when the data sources to be integrated are distributed, heterogeneous, and high in number. One recent solution to the issues of distribution and scale is to perform data integration using peer-to-peer (P2P) networks. Current P2P data integration architectures have mostly been flat, only specifying mappings directly between peers. Some do form the schemas into hierarchies, but none provide any abstraction of the schemas. This paper describes a set of general purpose P2P meta-data and data exchange primitives provided by an extended version of the AutoMed toolkit, and uses the primitives to implement a new architecture called iXPeer. iXPeer deals with integration on several levels of abstraction, where the lower levels define precise mappings between data source schemas, but the higher levels are loser associations based on keywords.

Keywords: P2P, data integration, model management, mapping discovery

1. INTRODUCTION

The database community has developed a number of approaches and systems to perform data integration in heterogeneous and distributed environments. In the **federated database** [18] approach, a user application would use a **global schema** built from mappings from **source schemas**, each source transformed to be represented in a single **common data model** (**CDM**). The global schema and the mappings are provided by a single software agent. The **mediator** approach [19], was intended to be more flexible. Each mediator is a separate agent and mediators are able to integrate information either directly from data sources or from other mediators to form a **mediated schema**. The AutoMed framework [11, 3] provides an implementation of the mediator approach that allows bi-directional mappings to be specified between data sources and the mediator. Whilst the mediator approach has been successful, the Internet now makes available many more data sources, stored in a wide range of data models, some of which are **dynamic** in the sense that one or more of their schema, availability, and location are subject to relatively frequent changes. The scale, heterogeneity, and dynamic nature make it impractical to build global/mediated schemas for user applications that are a simple hierarchical integration of data sources in a single modelling language. This has lead researchers to investigate a data model based **peer to peer** (**P2P**) [6, 4, 9] approach, where there are no strict hierarchies of mediators, but instead data exchange occurs directly between peers in the network. The end result is **P2P database management systems** (**PDMSs**).

WWW2006, May 22–26, 2006, Edinburgh, UK.
.

Many current approaches to PDMS, such as Piazza [6] and coDB [4] rely entirely to making mappings directly between peers, and have no further structuring of the network. All current PDMSs also work with a single CDM, for example, forcing a number of relational peers to translate their schemas in XML, causing unnecessary complication in the integration and query exchange process. We argue that using only direct P2P mappings is too complex. In particular, in any large scale P2P system, establishing complete mappings between all peers that might wish to communicate is an unfeasible proposition.

The work in this paper develops upon one approach to model based P2P data integration called **XPeer** [2] (different from the XPeer found in [17]). In XPeer there is a logical network organisation based on data semantics, which guides how the actual query processing and data exchange occurs between peers. Specifically, a data source peer can be promoted to one of the following three roles:

1. **cluster peer**: a mediator for a cluster of data source peers, which share related schemas (*e.g.* covering information about computing students). The mediated schema is called **cluster schema**. Each data source peer schema is defined as a view over the cluster schema using LAV mapping rules [5]. The cluster peer provides direct P2P mappings between its data source peers, which we have already argued that it is not scalable. Hence the following two additional levels above the cluster peer are introduced.

2. **domain peer**: a domain is a set of clusters sharing the same category of information (*e.g.* information about the students in all departments). The domain peer provides a simple directory service over cluster peers, locating which cluster peers support which set of keywords. These keywords are provided by the cluster peers when they publish their cluster schema onto their domain peer.

3. **global peer**: is an entry point for the system, locating where the domain peers are. A domain peer registering with the global peer provides a set of keywords that identify the category of information covered in its

domain.

The motivation behind this structure is that it reduces the amount of complete mappings that need to specified between data source peers, thus increasing the scalabitility of the architecture. In addition, the domain peers and the global peer are integrated at a higher level of abstraction (namely using just keywords) than the cluster peers and the data source peers (which are integrated using standard data integration mapping rules). As a result, data source peers can easily enter and exit the P2P network with only their cluster network being affected, thus making the architecture more flexible. However, assuming one global peer, one level of domain peers, and a single domain for each cluster peer, makes the P2P network liable to failure when a single domain peer or the global peer fails. In this paper we seek to remedy these problems, and implement XPeer into an extensible architecture **iXPeer**, readily adaptable to new applications. Specific contributions of this paper are:

- We describe in Section 3 the AutoMed P2P primitives, which give a flexible toolkit with which to implement model based P2P data integration. The toolkit does not prescribe a fixed approach to how the P2P data integration is performed, but rather methods that are of very general purpose, and could be used to implement a number of P2P architectures. Also, since AutoMed handles any structured or semi-structured data model [10], using AutoMed also makes our P2P system similarly flexible. The current implementation of AutoMed already handles relational, XML, ER, RDF, and semi-structured text files such as CSV.
- We extend in Section 4 the XPeer architecture into iXPeer, which is implemented using the AutoMed P2P primitives. In particular, we develop the iXPeer system so that it has a more flexible hierarchy, with any number of levels of domain peers, and the ability to join multiple hierarchies.

  The use of the AutoMed P2P primitives also has the advantage that we may further develop the iXPeer architecture in the light of experience gained in trails of P2P data integration.

Before describing the AutoMed P2P primitives, we give a new presentation of the AutoMed approach to data integration, called **both as view** (**BAV**) [11], that facilitates the description of P2P primitives in a concise manner. Comparison to related work will be left to Section 5, and our summary and conclusions are given in Section 6.

2. BAV AND THE AUTOMED TOOLKIT

A BAV meta database may be represented by a tuple: $\langle Schemas, Trans, SchemaObjs, AM\rangle$
where members of *Schemas* are linked with **transformation**s which are members of *Trans* and describe the mapping between two schemas. The function *SchemaObjs* applied to a schema returns the set of **schema objects** that represent the various constructs in the schema. The function AM applied to a schema will return a (possibly empty) set of **access methods** $\langle UserName, Password, URL, Driver\rangle$ that describe the data source that the schema wraps.

1. Each member of *Trans* is a tuple of one of the following kinds:
   - $\langle \mathsf{add}, \mathsf{construct}, SO, S_1, S_2, Q\rangle$: S_2 differs from S_1 in having an additional schema object SO of type construct, and the extent of SO is defined by query Q over S_1.
   - $\langle \mathsf{delete}, \mathsf{construct}, SO, S_1, S_2, Q\rangle$ is the reverse of add, and is equivalent to $\langle \mathsf{add}, \mathsf{construct}, SO, S_2, S_1, Q\rangle$
   - $\langle \mathsf{extend}, \mathsf{construct}, SO, S_1, S_2, \mathsf{Range}\ Q_1\ Q_2\rangle$: S_2 differs from S_1 in having an additional schema object SO, the extent of contains at least Q_1 and at most Q_2
   - $\langle \mathsf{contract}, \mathsf{construct}, SO, S_1, S_2, \mathsf{Range}\ \mathsf{Q}_1\ \mathsf{Q}_2\rangle$ is the reverse of extend, equivalent to $\langle \mathsf{extend}, \mathsf{construct}, SO, S_2, S_1, \mathsf{Range}\ \mathsf{Q}_1\ \mathsf{Q}_2\rangle$.
   - $\langle \mathsf{rename}, \mathsf{construct}, SO_1, SO_2, S_1, S_2\rangle$: S_2 differs from S_1 in that SO_1 in S_1 is renamed to SO_2 in S_2, and the extent of SO_1 equals SO_2.
   - $\langle \mathsf{ident}, S_1, S_2, \mathsf{function}\rangle$, the S_1 and S_2 have the same set of schema objects, and query processing may use function to combine the extents of S_1 and S_2 together [8].
2. A schema $S_x \in Schemas$ for which $AccessMethod(S_x)$ is non-empty is termed a data source schema.
3. A **BAV network** is a subset of *Schemas* which has the property that the transformations between the schemas form a connected graph, where there are no transformations that connect a member of the network to a non-member of the network.
4. A **pathway** $PW_{x,y} \subseteq Trans$ between S_x and S_y links a chain of schemas within a BAV network. Note that it has the property that every schema appearing in PW will appear in exactly two transformations, apart from S_x and S_y that appear in only one transformation.

EXAMPLE 1. **A meta database for two data sources: Computer Science Department ($\mathsf{S_{stu}}$) and University's Registry ($\mathsf{S_{reg}}$)**
$\mathsf{S_{stu}}$ contains the schema of the cs_student table in Figure 1 and $\mathsf{S_{reg}}$ the schema of student. At present there are no transformations in the repository.
$Schemas = \{\mathsf{S_{stu}}, \mathsf{S_{reg}}\}$
$Trans = \{\}$
$SchemaObjs(\mathsf{S_{stu}}) = \{\langle\langle \mathsf{cs\_student}\rangle\rangle, \langle\langle \mathsf{cs\_student,name}\rangle\rangle, \langle\langle \mathsf{cs\_student,term\_address}\rangle\rangle, \langle\langle \mathsf{cs\_student,year}\rangle\rangle, \langle\langle \mathsf{cs\_student,level}\rangle\rangle\}$
$SchemaObjs(\mathsf{S_{reg}}) = \{\langle\langle \mathsf{student}\rangle\rangle, \langle\langle \mathsf{student,name}\rangle\rangle, \langle\langle \mathsf{student,dept}\rangle\rangle, \langle\langle \mathsf{student,level}\rangle\rangle, \langle\langle \mathsf{student,home\_address}\rangle\rangle\}$
$AM(\mathsf{S_{stu}}) = \{\langle$lab, lab, jdbc:oracle:thin://cs.eg.uk/eg1, oracle.jdbc.driver.OracleDriver$\rangle\}$
$AM(\mathsf{S_{reg}}) = \{\langle$pjm, secret, jdbc:postgresql://reg.eg.uk/eg2, org.postgresql.Driver$\rangle\}$
□

Data integration involves establishing a mapping between data sources. Figure 2 lists a pathway between $\mathsf{S_{stu}}$ and $\mathsf{S_{reg}}$, which could be added to the *Trans* of Example 1 to give an integrated pair of data sources, where schema $\mathsf{S_f}$ can be considered a global schema over $\mathsf{S_{stu}}, \mathsf{S_{reg}}$.

Note that from the definitions of transformations $PW_{x,y}$ is equivalent to $PW_{y,x}$. We can also write a directed form of the pathway, starting at either schema.

cs_student				result			student			
name	term_address	year	level	name	course	mark	name	dept	level	home_address
Mary	180 Queen's...	5	ug	Mary	DB	77	Mary	Comp	ug	235 Princess St...
John	42 Sterling Pl...	4	pg	Mary	OS	58	John	Comp	pg	24 Lawn Market...
Jane	59 Evelyn Gard...	1	pg	Paul	OS	45	Jane	Comp	pg	102 Andrew's...
Fred	30 Pembridg...	3	pg	Paul	OOP	99	Fred	Comp	pg	71 Cornmarket...
Paul	82 Old Brompt...	1	ug				Paul	Comp	ug	93 Park Row...
							Iain	Math	ug	58 Tower Bridg...

Figure 1: Some example tables present in various data sources

$PW_{stu,reg}$={ ⟨extend,Table,⟨⟨student⟩⟩,S_{stu},S_b,Range [{x} | {x} <− ⟨⟨cs_student⟩⟩] Any⟩,
⟨extend,Column,⟨⟨student,name⟩⟩,S_b,S_c,Range [{x, y} | {x, y} <− ⟨⟨cs_student, name⟩⟩] Any⟩,
⟨extend,Column,⟨⟨student,dept⟩⟩,S_c,S_d,Range [{x, 'Comp'} | {x} <− ⟨⟨cs_student⟩⟩] Any⟩,
⟨extend,Column,⟨⟨student,level⟩⟩,S_d,S_e,Range ⟨⟨cs_student, level⟩⟩ Any⟩,
⟨extend,Column,⟨⟨student,home_address⟩⟩,S_e,S_f,Range Void Any⟩
⟨contract,Column,⟨⟨cs_student,term_address⟩⟩,S_f,S_g,Range Void Any⟩,
⟨contract,Column,⟨⟨cs_student,year⟩⟩,S_g,S_h,Range Void Any⟩,
⟨delete,Column,⟨⟨cs_student,level⟩⟩,S_h,S_i,[{x, y} | {x, y} <− ⟨⟨student, level⟩⟩; {x, 'Comp'} <− ⟨⟨student, dept⟩⟩]⟩,
⟨delete,Column,⟨⟨cs_student,dept⟩⟩,S_i,S_j,[{x, y} | {x, y} <− ⟨⟨student, level⟩⟩; {x, 'Comp'} <− ⟨⟨student, dept⟩⟩]⟩,
⟨delete,Column,⟨⟨cs_student,level⟩⟩,S_j,S_k,[{x, y} | {x, y} <− ⟨⟨student, level⟩⟩; {x, 'Comp'} <− ⟨⟨student, dept⟩⟩]⟩,
⟨delete,Column,⟨⟨cs_student,name⟩⟩,S_k,S_l,[{x, y} | {x, y} <− ⟨⟨student, name⟩⟩; {x, 'Comp'} <− ⟨⟨student, dept⟩⟩]⟩,
⟨delete,Table,⟨⟨cs_student⟩⟩,S_l,S_{reg},[{x} | {x} <− ⟨⟨student⟩⟩; {x, 'Comp'} <− ⟨⟨student, dept⟩⟩]⟩}

Figure 2: A pathway linking S_{stu} and S_{reg}

The AutoMed **repository** [3] provides the basic storage mechanism for schemas and BAV pathways, around which a number of tools have been developed. Of relevance to the work in this paper are:

1. match_merge(S_x,S_y): takes a pair of schemas S_x,S_y and builds a BAV pathway between them. First, the schemas are *matched* to identify mappings between their objects and then the pathway is built based on the identified mappings [15]. One schema, S_z, within that pathway will contain all the information from S_x and S_y, and hence may be used as a **global schema** for querying the contents of S_x and S_y as a single data source.

2. reformulate_query(Q,S_x,S_1^D,...,S_n^D) [7]: takes a query Q on S_x, and reformulates it into a query over the data source schemas $S_1^D, \ldots, S_n^D$, where D here denotes a data source.

For example, the result of match_merge(S_{stu},S_{reg}) would be global schema S_f linked to S_{stu} with the pathway $PW_{stu,f}$ seen in Figure 2 and linked to S_{reg} with the reverse pathway of $PW_{f,reg}$, which would extend S_{reg} with the schema objects in S_{stu}. Performing the action:
reformulate_query(distinct [{x, y} |
{x, y} <− ⟨⟨student, dept⟩⟩],S_f,S_{stu},S_{reg})
on S_f results in
distinct [{x, y} | [{x, 'Comp'} | {x} <− S_1 : ⟨⟨cs_student⟩⟩];
{x, y} <− S_{reg} : ⟨⟨student, dept⟩⟩]

3. AUTOMED P2P PRIMITIVES

A BAV meta database explained in the previous section is stored in an AutoMed repository. Each AutoMed repository has two elements: a modelling language repository called the **MDR** and a schema and transformation repository called the **STR** [3]. The AutoMed P2P implementation (a Java API available from http://www.doc.ic.ac.uk/automed/) allows separate AutoMed repositories to communicate with each other, each repository acting as a **peer** in a P2P network. The peers may exchange meta-data information about schemas and pathways, and may also request the execution of queries on other peers. The architecture is illustrated in Figure 3. For the purposes of the current work, it is assumed that all modelling languages definitions used by peers will be known in advance, and do not need to be distributed on the P2P network.

Access to each AutoMed repository is made through an AutoMedPeer unique to that repository. Each AutoMedPeer has a unique name assigned based on the repository location. Potentially, several AutoMedPeers might be available on any one host, and so access to any AutoMedPeer is made via a P2PRegistry, of which one must execute on any host that runs an AutoMedPeer. This P2PRegistry runs on a port number that is fixed for the P2P network (number 8282 by default), which means any other peer wishing to communicate with an AutoMedPeer needs only the name and the IP address of the peer.

The IP address of an AutoMedPeer may be obtained from the P2PDirectory, that behaves as a meta-data directory service for the P2P network, providing the IP address of peers, together with basic meta data information about the peers. The P2PDirectory service is very simple, and as we will demonstrate in this paper can be arranged into a hierarchy to give a very similar operational structure to the DNS directory service used on the Internet to lookup IP addresses based on supplied domain names. We describe the primitives provided by the directory service in the next subsection, and then describe the meta data exchange and query exchange primitives in the following subsections.

3.1 The Directory Service

The directory service provides a minimal database (summarised in Figure 4) which stores in peer an instance for each peer on the P2P network, with its name and IP address, and in schema an instance for each **public schema** [12] that some peer repository chooses to make publicly available. Public schemas are first proposed by one peer, which

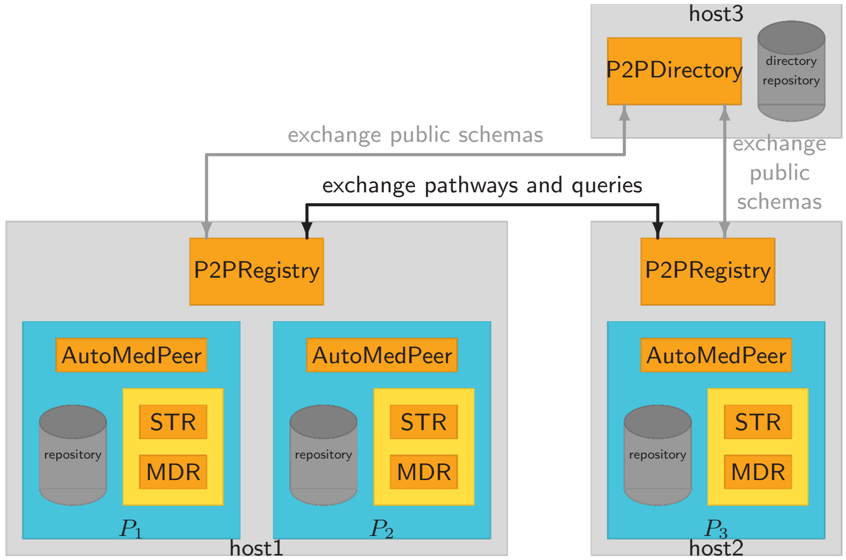

Figure 3: The AutoMed P2P Architecture

will send the details of the schema to the directory, and then any other peer may integrate its own data sources with the public schema and inform the directory that it has a pathway that links the public schema to its data sources. Hence, the directory stores a subset of the schemas that are present on all peers under the directory, together with a statement about which are able to connect that schema via a pathway to a data source, and hence service queries on that schema.

Instances of the peer entity are created and read by two AutoMed P2P primitives:

- $P_x.D$.register(): inform the directory D of the name and IP address of peer P_x, creating or updating an instance of peer. When any peer logins to the P2P network, it registers with its local P2PRegistry, and then the P2PDirectory. Thus the P2PDirectory will record the IP address of the P2PRegistry and the P2PRegistry records the local port number of the AutoMedPeer.

- $P_x.D$.lookup(P_y): lookup of IP address for another peer P_y from D.

Instances of the schema entity, and its relationship pathway with peer are created and read by the following primitives:

- $P_x.D$.publish_schema(S, $SchemaObjs(S)$, description): store a copy of S held in P_x in the directory D as an instance of schema, with a description and list of schema objects within the schema. From a logical perspective, this makes no change to the total number of schemas in the P2P network, but from an operational perspective, it increases the number of schemas which may be shared between peers.

- $P_x.D$.get_schema(S): retrieve a copy of S from the directory's schema, and add S with its schema objects to $Schemas_x$ and $SchemaObjs_x$ of P_x.

- $P_x.D$.advertise_pathway(S): update the directory with an an association between P_x and S to record that P_x stores a pathway from S to one or more data sources (and hence will be able to return answers to queries on S).

- $P_x.D$.get_peers_for_schema(S): return the set of peers $P_1, \ldots, P_n$ that have performed P_y.advertise_pathway(S), $y = \{1, \ldots, n\}$. This informs P_x which peers it may contact in order to obtain (1) answers to queries on S and (2) meta data about pathways from S to other schemas.

- $P_x.D$.get_schemas_for_peer(P_y): return the set of public schemas $S_1, \ldots, S_n$ that P_y has performed P_y.advertise_pathway(S_z), $z = \{1, \ldots, n\}$.

EXAMPLE 2. **Publishing part of data source**
Suppose peer $\mathsf{P_{cs}}$ of a university's computer science department holds a data source $\mathsf{S_{cs}}$ made up of the cs_student and result tables of Figure 1. It decides to make public only information about students and not their results. To do this, it must create a schema, say called $\mathsf{S_{stu}}$, which it can do by the pathway below:

$\mathsf{PW_{cs,stu}}$={⟨contract,Column,⟨⟨result,grade⟩⟩,$\mathsf{S_{cs}}$,$\mathsf{S_{csa}}$,Range Void Any⟩,
⟨contract,Column,⟨⟨result,course⟩⟩,$\mathsf{S_{csa}}$,$\mathsf{S_{csb}}$,Range Void Any⟩,
⟨contract,Column,⟨⟨result,name⟩⟩,$\mathsf{S_{csb}}$,$\mathsf{S_{csc}}$,Range Void Any⟩,
⟨contract,Table,⟨⟨result⟩⟩,$\mathsf{S_{csc}}$,$\mathsf{S_{stu}}$,Range Void Any⟩}

The peer then invokes $\mathsf{P_{cs}.D_1}$.publish_schema($\mathsf{S_{stu}}$), causing $\mathsf{S_{stu}}$ to be added into the directory. It then issues $\mathsf{P_{cs}.D_1}$.advertise_pathway($\mathsf{S_{stu}}$) to indicate that it can answer queries on $\mathsf{S_{stu}}$ (since peer $\mathsf{P_{cs}}$ has pathway $\mathsf{PW_{cs,stu}}$ relating that schema to data source $\mathsf{S_{cs}}$). □

3.2 P2P Meta Data Exchange

The P2P meta data exchange primitives allow peers to exchange pathways to any associated schemas and between each other. The two primitives available are:

- $P_x.P_y$.get_source_pathways(S): obtain from P_y all pathways held at P_y from public schema S to any data sources $S_1^D, S_2^D, \ldots$, along with $AM(S_1^D)$, $AM(S_2^D), \ldots$

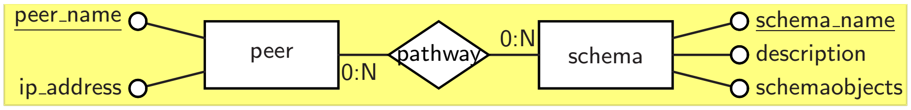

Figure 4: P2P Directory Schema

and store them in P_x. This means that P_x gains the knowledge from P_y of how to map queries between S and $S_1^D, S_2^D, \ldots$, and how to access the associated databases, such that in future it can use these data sources without any intervention of P_y. In particular P_x becomes able to execute queries over $S_1^D, S_2^D, \ldots$ without the aid of P_y. P_y is at liberty to refuse this primitive's request if it does not wish to divulge this information.

- $P_x.P_y.\mathsf{get\_pathway}(S_i, S_j)$: update the repository of P_x with a copy of the pathway from S_i to S_j held at P_y.

Example 3. **Revealing Data Sources**
Peer $\mathsf{P_{reg}}$ holding $\mathsf{S_{reg}}$ decides to integrate itself with the already published schema $\mathsf{S_{stu}}$ on $\mathsf{P_{cs}}$ using the following steps:

- $\mathsf{P_{reg}.D.get\_schema(S_{stu})}$: obtain from the directory a copy of $\mathsf{S_{stu}}$ and put it in the repository of $\mathsf{P_{reg}}$.
- $\mathsf{P_{reg}.D.get\_peers\_for\_schema(S_{stu})}$: obtain the peers that implement $\mathsf{S_{stu}}$, *i.e.* peer $\mathsf{P_{cs}}$.
- $\mathsf{P_{reg}.P_{cs}.get\_source\_pathways(P_{cs}, S_{stu})}$: obtain a copy from $\mathsf{P_{cs}}$ of the pathways from $\mathsf{S_{stu}}$ to any data source schemas. In this case just $\mathsf{S_{cs}}$ with its access method is copied into the repository at $\mathsf{P_{reg}}$.
- $\mathsf{P_{reg}.match\_merge(S_{reg}, S_{cs})}$: $\mathsf{P_{reg}}$ merges its own data source $\mathsf{S_{reg}}$ with the data source schema of $\mathsf{P_{cs}}$ and forms a single BAV network which may then be queried. This means $\mathsf{P_{reg}}$ is able to access the data sources of $\mathsf{P_{cs}}$ without needing to communicate with $\mathsf{P_{cs}}$.

□

3.3 P2P Query Processing

The P2P meta data exchange primitives allow a peer to gather meta data from other peers, and assemble its own integration of data sources. However, peers may not be willing to divulge all their meta data information, and also maintaining a copy of the meta data will cause problems in large and evolving P2P networks. Hence an alternative interaction between peers is for queries to be sent to remote peers for evaluation. The AutoMed P2P query exchange primitives allow queries to be distributed over the P2P network as follows:

1. $P_x.P_y.\mathsf{evaluate\_query}(Q, S)$: execute query Q on schema S of peer P_y, and return to P_x the results.
2. $P_x.P_y.\mathsf{evaluate\_broker\_query}(Q, S)$: perform
   $P_x.P_y.\mathsf{evaluate\_query}(Q, S)$
   but in addition, request P_y to perform
   $P_y.P_z.\mathsf{evaluate\_broker\_query}(Q, S)$
   on each peer P_z that P_y knows to implement S (other than P_x and any other peers the broker query has passed through).
3. $P_x.D.\mathsf{wrap\_public\_schema}(S, P_y)$: perform
   $P_x.D.\mathsf{get\_schema}(S)$
   and treat S as a data source in P_x, executing
   $P_x.P_y.\mathsf{evaluate\_query}(Q, S)$.

Example 4. **Restricting access to meta-data about data sources**
In Example 3, $\mathsf{P_{cs}}$ accepted the request
$\mathsf{P_{reg}.P_{cs}.get\_source\_pathways(P_{cs}, S_{stu})}$
however in some other case it might not want to release details of its data sources, hiding the schemas and access methods from other peers, but still allowing some query processing to occur. In this case $\mathsf{P_{reg}}$ could:

- Issue a $\mathsf{P_{reg}.D_1.wrap\_public\_schema(S_{stu}, P_{cs})}$, which would create a copy of $\mathsf{S_{stu}}$ in the repository of $\mathsf{P_{reg}}$, with an access method pointing at $\mathsf{P_{cs}}$ of the form
  $\langle \mathsf{P_{cs}}, \text{“”}, \mathsf{P_{reg}}, \mathsf{P2PDirectory} \rangle$.
  Note that the $\mathsf{P2PDirectory}$ is there as the source, since it will be used to resolve the location of its 'user' $\mathsf{P_{cs}}$ at runtime, allowing peers to be mobile and change IP addresses.
- Perform $\mathsf{P_{reg}.match\_merge(S_{reg},S_{stu})}$ to build in the repository of $\mathsf{P_{reg}}$ a pathway $\mathsf{PW_{stu,reg}}$ between $\mathsf{S_{reg}}$ and $\mathsf{S_{stu}}$.
- Performing Q'=$\mathsf{P_{reg}.reformulate\_query}(Q,S_x,\mathsf{S_{stu}})$ on any schema S_x in the pathway $\mathsf{PW_{stu,reg}}$ will return Q' to send to $\mathsf{P_{cs}}$. This query is sent using
  $\mathsf{P_{reg}.P_{cs}.evaluate\_query}(Q', \mathsf{S_{stu}})$
  When $\mathsf{P_{cs}}$ accepts this request it will use
  $Q''= \mathsf{P_{cs}.reformulate\_query}(Q',\mathsf{S_{stu}},\mathsf{S_{cs}})$
  to obtain query Q'' to execute on $\mathsf{S_{cs}}$.

□

4. BUILDING IXPEER USING AUTOMED P2P PRIMITIVES

We now present a revision of the XPeer architecture called **iXPeer** built around the AutoMed P2P primitives. As outlined in Figure 5, we remove the distinction between domain peer and global peer, and allow domain peers to form any hierarchical structure. Also, though not illustrated, the domain peers and cluster peers are allowed to participate in any number of hierarchies.

4.1 Establishing the P2P Network

Using AutoMed, a low level peer in the iXPeer architecture is any data source that is willing to give access to its information through its access method *AM*. A cluster peer is a single AutoMedPeer, that stores a BAV meta database, *i.e.* a BAV network of all the schemas in the cluster, together with their access method details. Hence in Figure 5, we show the cluster peers $C_1, C_2, \ldots$ comprising of a number of schemas $S_1, S_2, \ldots$ connected to data sources via access methods. Each cluster peer has a **cluster schema** that makes public to the P2P network and the other peers. For example, in Figure 5, the cluster schema of cluster peer C_1 is S_7^C.

A domain peer is both a P2PDirectory and an AutoMedPeer. As a P2PDirectory it allows cluster peers to publish their cluster schemas. A cluster peer C_i with cluster schema S_i^C that wants to connect to a domain peer D_j needs to perform the following steps:

1. $C_i.D_j.\mathsf{register}()$
2. $C_i.D_j.\mathsf{publish\_schema}(S_i^C, SchemaObjs(S_i^C), Desc_i^C)$
3. $C_i.D_j.\mathsf{advertise\_pathway}(S_i^C)$

The domain peer as an AutoMedPeer publishes its P2P directory schema (Figure 4) in its P2PDirectory and thus its cluster peers. The P2P directory schema associates the cluster schemas in the domain with their descriptions and it is necessary for the cluster peers to create their **XPeer view** (more on this in Section 4.2). The following steps are executed by each domain peer D_j with P2P directory schema DS_j^D:

1. $D_j.D_j.\mathsf{register}()$
2. $D_j.D_j.\mathsf{publish\_schema}(DS_j^D, SchemaObjs(DS_j^D), Desc_j^D)$
3. $D_j.D_j.\mathsf{advertise\_pathway}(DS_j^D)$

For a domain peer to inform other domain peers in the P2P network about cluster schemas in its domain, it must advertise a pathway from its P2P directory schema to the P2P directory schema of a domain peer in a higher level in the iXPeer architecture. Suppose that D_k is a domain peer one level above D_j with a public P2P directory schema DS_k^D (*i.e.* the schema in Figure 4), then the steps that D_j needs to execute are:

1. $D_j.D_k.\mathsf{register}()$
2. $D_j.D_k.\mathsf{wrap\_public\_schema}(DS_k^D, D_k)$
3. $D_j.\mathsf{match\_merge}(DS_j^D, DS_k^D)$
4. $D_j.D_k.\mathsf{advertise\_pathway}(DS_k^D)$

Note that iXPeer extends the previous XPeer model in allowing domain peers to form arbitrarily nested hierarchies. For example, in Figure 5, D_2 has registered with it both a cluster peer C_3 and another domain peer D_1. Also D_1 is free to register with another domain peer D_3, thus joining multiple hierarchies. The actual topology of the P2P network, *e.g.* the number of domain peers used as roots, the levels of domains, *etc*, depends on the particular application and the requirements of the PDMS being built and it can always be adjusted using the AutoMed P2P primitives. iXPeer is free of single points of failure since any peer in the network can be associated with more than one other peers. Note that we are not examining failure recovery in this paper. Depending on the application and the recovery approach required, data and meta data can be cached in different peers, cluster peers can be registered with more than one domain peer, *etc*.

4.2 Querying and Building Applications over the P2P Database

Querying semantically related data sources depends on the ability to map between their schemas. Unfortunately, in most cases matching between schemas is still largely performed manually or semi-automatically. P2P based related work (*e.g.* SomeWhere, Piazza) have assumed that the sources (peers) are able to declare their mappings with at least another source (peer) of the P2P network. This constraint imposed on the peers could be raised if the system is able to automatically discover semantic mappings between the sources starting from their ontologies or schemas.

In iXPeer, a client wishing to query or develop applications over the distributed information system is provided with a mediated schema called an **XPeer view** over relevant cluster schemas. As argued by [16], the existence of a unified view over heterogeneous data sources makes easier the development of applications. In iXPeer **view creation** is conducted as follows:

1. A cluster peer poses a meta data query on the P2P directory schema of the domain peer which the cluster peer is registered. This query is formed of a set of keywords $K_q = \{K_1, \ldots, K_N\}$, which describe the area of interest of the cluster peer. The purpose of this query is the identification of all cluster schemas in the P2P network that are related with the specific set of keywords. Suppose that C is a cluster peer registered to a domain peer D_j with DS_j^D as P2P directory schema. The cluster peer performs the following step:

   $C.D_j.\mathsf{evaluate\_query}([\{p, ip, s, kw\} \mid$
   $\{p, ip\} <\!- \langle\langle\mathsf{peer,ip\_address}\rangle\rangle;$
   $\{p, s\} <\!- \langle\langle\mathsf{pathway,peer,schema}\rangle\rangle;$
   $\{s, kw\} <\!- \langle\langle\mathsf{schema,description}\rangle\rangle;$
   $\{kw\}\ \mathsf{contains}\ \{K_1, \ldots, K_N\}], DS_j^D)$

   This action is performed on any (or all) domain peers to which the cluster peer is registered.

2. A domain peer responds to this request by first evaluating the above query on its own repository. If there is a set of cluster schemas CSs in its domain that satisfy all the keywords in the query then the domain peer returns this set as the answer to the query. If there are keywords which are not covered then the domain peer forwards the query to other domain peers higher in the XPeer hierarchy by performing the following action:

   $D_j.D_j.\mathsf{evaluate\_broker\_query}([\{p, ip, s, kw\} \mid$
   $\{p, ip\} <\!- \langle\langle\mathsf{peer,ip\_address}\rangle\rangle;$
   $\{p, s\} <\!- \langle\langle\mathsf{pathway,peer,schema}\rangle\rangle;$
   $\{s, kw\} <\!- \langle\langle\mathsf{schema,description}\rangle\rangle;$
   $\{kw\}\ \mathsf{contains}\ \{K_1, \ldots, K_N\}], DS_j^D)$

   The same process is followed by the other domain peers higher in the hierarchy. Ideally, we would want to control this query propagation, for example stopping once all keywords are covered, or covered a certain number of times; the decision depends on the requirements of the PDMS being built. At present the query distribution aspects of the AutoMed toolkit are being developed to support such query distribution rules.

3. The result of these queries is a set of tuples that contain the peer name, peer ip_address, public schema and associated keywords of cluster peers that are related to the initial set of keywords K_q. The cluster peer that initiated the query may therefore execute a process similar to that in Example 4, wrapping peer schemas it considers (from information in the keywords and possibly previous experience of using a peer) relevant, and

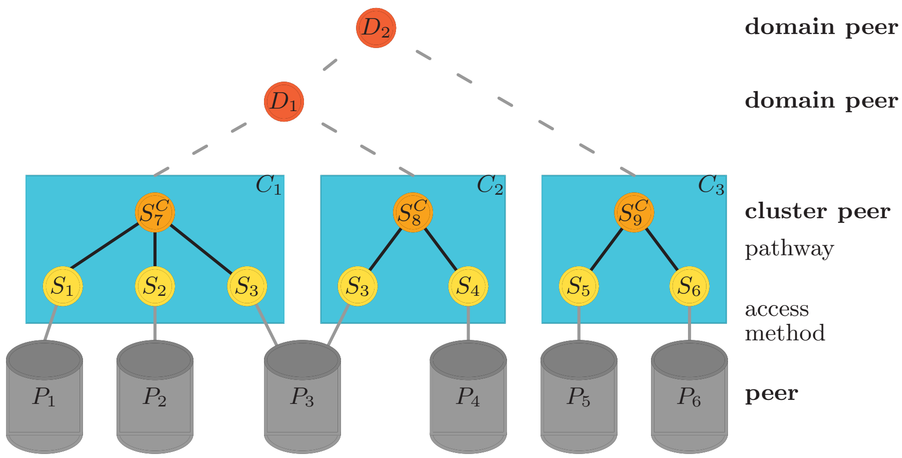

Figure 5: iXPeer: Revised XPeer architecture using AutoMed P2P primitives

using the match_merge primitive to create a mediated schema called the **XPeer view**.

Example 5. **Creating and using an XPeer view**
Consider that the P2P network shown in Figure 5 is used by a university's P2P database. Let cluster peer C_1 be the peer in the Computer Science Department of the university and C_2 the peer in the Maths Department. These two cluster peers belong to the same sub-domain represented by domain peer D_1. Cluster peer C_3 is the peer in the university's Registry. Domain peer D_2 is the high-level peer which represents the university.

The Computer Science Department peer C_1 publishes on D_1 its cluster schema $\mathsf{S_{stu}}$ described in Example 2 which represents information about the computer science department's students. The description of published schema $\mathsf{S_{stu}}$ contains keywords $\{\mathsf{computing}, \mathsf{student}, \mathsf{marks}\}$. The Maths Department peer C_2 publishes a similar cluster schema, $\mathsf{S^P_{math}}$, on D_2 about mathematics students, with a description that contains keywords $\{\mathsf{maths}, \mathsf{student}, \mathsf{marks}\}$. The Registry peer C_3 has cluster schema $\mathsf{S_{reg}}$, as described in Example 1, with information about all the students in the university and publishes it on peer D_1. The description of $\mathsf{S_{reg}}$ contains keywords $\{\mathsf{students}, \mathsf{home\_address}\}$.

Now, the Computer Science Department peer wants to learn the home addresses of its graduate students to send them their degrees certificates and congratulate them. First it needs to create its XPeer view by performing:

$C_1.D_1.\mathsf{evaluate\_query}([\{p, ip, s, kw\} \mid$
$\quad \{p, ip\} <\!\!- \langle\langle \mathsf{peer,ip\_address} \rangle\rangle;$
$\quad \{p, s\} <\!\!- \langle\langle \mathsf{pathway,peer,schema} \rangle\rangle;$
$\quad \{s, kw\} <\!\!- \langle\langle \mathsf{schema,description} \rangle\rangle;$
$\quad \{kw\}\ \mathsf{contains}\ \{\mathsf{computing}, \mathsf{home\_address}\}], DS_1^D)$

Peer D_1 is not able to answer this query based on its own repository since schemas $\mathsf{S_{stu}}$ and $\mathsf{S^P_{math}}$ do not cover these keywords. Therefore, it will perform

$D_1.D_1.\mathsf{evaluate\_broker\_query}([\{p, ip, s, kw\} \mid$
$\quad \{p, ip\} <\!\!- \langle\langle \mathsf{peer,ip\_address} \rangle\rangle;$
$\quad \{p, s\} <\!\!- \langle\langle \mathsf{pathway,peer,schema} \rangle\rangle;$
$\quad \{s, kw\} <\!\!- \langle\langle \mathsf{schema,description} \rangle\rangle;$
$\quad \{kw\}\ \mathsf{contains}\ \{\mathsf{computing}, \mathsf{home\_address}\}], DS_1^D)$

This query would follow pathways that are connected with other domain peers in the P2P network. When the domain peer D_2 receives the query it will be able to respond to it since (a) it has a published schema $\mathsf{S_{reg}}$ from C_3 which covers keyword home_address and (b) it has a subdomain D_1 which has a published schema $\mathsf{S_{stu}}$ that covers keyword computing. Therefore D_2 is going to respond by sending back the IP addresses of C_3 and C_1 and their schemas $\mathsf{S_{reg}}$ and $\mathsf{S_{stu}}$.

Peer C_1 will now perform match_merge($\mathsf{S_{reg}}$,$\mathsf{S_{stu}}$) that will create the global schema S_f of Figure 2. This schema will be the XPeer view of C_1:

cs_student			
name	term_address	year	level

student			
name	dept	level	home_address

Now C_1 just has to execute on $\mathsf{S_f}$ the query:

$\mathsf{reformulate\_query}([\{\mathsf{x}, \mathsf{y}\} \mid \{\mathsf{s}, \mathsf{x}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{name} \rangle\rangle;$
$\quad \{\mathsf{s}, \mathsf{w}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{year} \rangle\rangle;$
$\quad \{\mathsf{s}, \text{‘ug’}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{level} \rangle\rangle; \mathsf{w} > 4;$
$\quad \{\mathsf{s}, \mathsf{y}\} <\!\!- \langle\langle \mathsf{student}, \mathsf{home\_address} \rangle\rangle])$

that will result in the evaluation of the query

$[\{\mathsf{x}\} \mid \{\mathsf{s}, \mathsf{x}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{name} \rangle\rangle;$
$\quad \{\mathsf{s}, \mathsf{w}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{year} \rangle\rangle;$
$\quad \{\mathsf{s}, \text{‘ug’}\} <\!\!- \langle\langle \mathsf{cs\_student}, \mathsf{level} \rangle\rangle; \mathsf{w} > 4]$

on C_1's local data sources, and performing the action

$C_1.C_3.\mathsf{evaluate\_query}(\langle\langle \mathsf{student,home\_address} \rangle\rangle, \mathsf{S_{reg}})$ □

5. RELATED WORK

There are several P2P file sharing systems such as Gnutella, Napster, Morpheus, *etc*, with different degrees of centralisation. The main difference between the file sharing P2P systems and P2P database systems is that the P2P database management systems (PDMSs) are model based, using a schema based on the model to structure the data they contain, and providing a query language for that model. We divide these PDMSs into those which are designed to build upon the semantic web, and those extending the traditional data mappings.

Edutella [13] is a semantic web oriented project implementing an RDF-based metadata infrastructure for P2P networks to enable interoperability between heterogeneous JXTA applications. The overlay network underlying Edutella is a hypercube of super-peers to which peers are directly connected. Each super-peer acts as a mediator and it routes the query to some of its neighbour super-peers according to a strategy exploiting the hypercube topology for guaranteeing a worst-case logarithmic time for reaching the relevant super-peer. SomeWhere [1] is semantic web oriented P2P mediation based on propositional logic for defining ontologies, mappings, and queries.

Turning to review data model mediation systems, in which category iXPeer falls, the Piazza [6] system focused on a specification language for describing the semantic mappings between the peers. More precisely, the main goal was to enable users to specify mappings between a small set of peers using peer descriptions instead of a mediated schema. This language also enables the description of each peer. PeerDB [14] is a project implementing relational model P2P database functions, such as automatic schema matching. Another system named XPeer [17] has an architecture for XML data integration with two kinds of nodes: peers and super-peers. All the super-peers having the same parent form a group. Super-peers are organised to form a tree where each node owns schema information about its children. The super-peer schema is built simply as the union of those of its children.

There are many differences between our approach and that of other PDMSs. For example, Piazza does not provide a mediated schema but mappings between a small set of peers. A peer is assumed to be connected into a small set of peers, and by iteration, it can reach many peers. This solution is prone to degradation of query performance. Like Piazza, SomeWhere [1] has assumed that each peer is able to declare mappings between its ontology and the ontologies of some peers that it knows. In our system, a peer joining the P2P network need only provide some keywords that will be used to identify which clusters it should belong to. The approach closest to ours is [17], but this is limited to forming strict segregated hierarchies, and super-peers contain all the schema information of sub-peers.

6. SUMMARY AND CONCLUSIONS

We have presented an extension of the AutoMed toolkit that permits the exchange of meta-data and queries between peers on a P2P network, and we have shown how this extended toolkit may be used to implement a novel architecture called iXPeer. The iXPeer architecture allows an arbitrarily structured hierarchy to be built between domain peers, and domain peers to join several hierarchies, thus giving the overall P2P network resilience in the face of failures at peers. Also, basing iXPeer on AutoMed means that we built a P2P network capable of handling any structured or semi-structured data model.

The AutoMed toolkit is very flexible in what it permits to be built, and hence basing iXPeer around this toolkit allows us to adapt and improve the iXPeer architecture in the light of our on going experimental work currently being conducted. Current work is focusing on query distribution control, which will be used to control the processing of searching for keywords matching when processing a keyword query during XPeer view generation.

7. REFERENCES

[1] P. Adjiman, P. Chatalic, F. Goasdoué, M-C. Rousset, and L. Simon. Somewhere in the semantic web, 2005.

[2] Z. Bellahsène and M. Roantree. Querying distributed data in a super-peer based architecture. In *Proc. DEXA 2004*, volume 3180 of *LNCS*, pages 296–305, 2004.

[3] M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE'04*, volume 3084 of *LNCS*, pages 82–97. Springer, 2004.

[4] E. Franconi, G. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB robust peer-to-peer database system. In *Proc. 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing*, 2004.

[5] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[6] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proc. WWW'03*, 2003.

[7] E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL queries and migrating data in the AutoMed toolkit. Technical Report No. 20, AutoMed, 2003.

[8] E. Jasper, N. Tong, P.J. McBrien, and A. Poulovassilis. View generation and optimisation in the AutoMed data integration framework. In *Proc. Baltic DB&IS04*, volume 672 of *Scientific Papers*, pages 13–30. Univ. Latvia, 2004.

[9] A. Kementsietsidis. Data sharing and querying for peer-to-peer data management systems. In *EDBT PhD Workshop*, pages 177–186, 2004.

[10] P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99*, volume 1626 of *LNCS*, pages 333–348. Springer, 1999.

[11] P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238. IEEE, 2003.

[12] P.J. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Proc. DBISP2P, at VLDB'03*, pages 91–107, 2003.

[13] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. Query language for semantic web: EDUTELLA: a P2P networking infrastructure based on RDF. In *Proc. 11th WWW Conf.*, 2002.

[14] B.C. Ooi and K-L. Tan Y. Shu. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.

[15] N. Rizopoulos and P.J. McBrien. A general approach to the generation of conceptual model transformations. In O. Pastor and J.F. e Cunha, editors, *Proc. CAiSE'05*, volume 3520 of *LNCS*, pages 326–341. Springer, 2005.

[16] 1999 S. Abiteboul. On views and XML. In *Proc. PODS*, pages 1–9, 1999.

[17] C. Sartiani, P. Manghi, G. Ghelli, and G. Conforti. XPeer: A self-organizing XML P2P database system. In *Proc. P2P&DB Workshop*, pages 456–465, 2004.

[18] A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, 22(3):183–236, 1990.

[19] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.