Mapster: A Peer-to-Peer Data Sharing Environment

Colin Rouse
Department of Computer Science
University of Cape Town
Western Cape
South Africa
crouse@cs.uct.ac.za

Prof. Sonia Berman
Department of Computer Science
University of Cape Town
Western Cape
South Africa
sonia@cs.uct.ac.za

Abstract

This paper describes a system called Mapster that allows users in a P2P network to share their databases. The research addresses problems of heterogeneity and scalability in P2P databases. To provide fine-grained access to users' databases, schema matching and a super-peer topology are used. The schema matching component allows information to be translated by semi-automatically determining the mappings between the databases within the P2P network. A super-peer topology enables the schema matching techniques to operate effectively in large, dynamic, heterogeneous networks.

Keywords:

Peer-to-peer, database sharing, schema matching, super-peer, JXTA

Computing Review Categories:

H.2.4, H.3.3, H.3.4

1 Introduction

Much recent research and development has focused on aspects of peer-to-peer (P2P) systems, including resource-sharing [15], searching [26], and security [5]. Search has received a great deal of attention, particularly in file-sharing P2P systems. By extending the search capability from files to databases, a deeper level of information sharing can be achieved.

A key success of P2P systems has come from the ease of use they offer: a user can connect to the network, access or query a particular resource and then simply leave. This flexibility helped make file-sharing systems such as Napster [28] highly successful. For peer-to-peer database (P2PDB) systems to be successful, they need to be able to communicate seamlessly with other peers' databases, and to support querying and data transfer with little user effort.

Much human information resides in databases, which support fine-grained requests for specific subsets of their data. A P2PDB system permits querying a network of databases that are independently designed and managed. It offers the benefits of database access as well as the advantages of P2P communication. Data sharing in this manner is more flexible than Web-based systems, and avoids problems of centralised control inherent in client-server or multidatabase systems. This paper describes Mapster [24], our P2PDB system which aims to provide fine-grained sharing of heterogeneous relational databases in a manner that can scale well. As part of the solution, a schema interoperability approach that will operate effectively in P2P environments is needed. Our approach takes advantage of a P2P overlay network structure based on super-peer clusters [24].

Schema interoperability is required for databases to communicate with each other. This can be achieved using semantic mappings between databases. Schema interoperability is not new, but only a few systems, viz. Piazza [12], Hyperion [1] and BestPeer [22], have implemented some form of it within a P2PDB network. We exploit a specific network topology, namely super-peer clusters, to make this viable in dynamic P2P networks.

The rest of the paper is organized as follows. Sections 2 and 3 briefly cover the background of P2P systems and schema matching, respectively, and section 4 gives an overview of the Mapster architecture. Sections 5 and 6 describe interoperability and query processing, and the components of the prototype system are presented in section 7. Experimental results are discussed in section 8. Section 9 compares Mapster with related work. The last sections provide a conclusion and suggestions for future work.

2 P2P Networks

P2P systems are applications that allow a network of peers to produce or consume a variety of resources in a scalable and efficient manner [31]. Peers may range from cell phones to high-end servers. These peers have control over their own resources and choose when to be connected to the network. In pure P2P networks, there are no central servers, all peers are considered equal and they connect directly with each other [14]. P2P systems are different to client-server systems, as there is little node specialisation, the peers are more interconnected and the workload is typically divided amongst the peers. The P2P architecture allows ad hoc networks to be created very easily and can connect existing databases without requiring them to be changed. Client-server applications tend to be less flexible and impose more rules and limitations upon participating nodes.

Napster[28], a simple yet effective P2P file-sharing system, was primarily responsible for popularising modern P2P systems. They are now a very active research area. This research has produced many systems that range Internet telephony networks like Skype [2], to file-sharing systems, like Gnutella [17], which are the most common application of the P2P architecture.

The large-scale, dynamic, heterogeneous characteristics of P2P systems pose three key challenges, viz. resource management, search, and security [7]. Resource management covers issues such as load balancing, fairness (which aims to prevent those that do not contribute from benefiting), and data replication. Search enhancement has focussed on routing algorithms [6], distributed hash tables (DHTs) [26], and caching [4]. More information on P2P file-sharing systems can be found in [4] and [7]. After describing Mapster, section 9 outlines existing P2P *database* systems and compares them with our approach.

2.1 Topologies

The P2P topology defines the layout of the underlying network and dictates how the peers connect to each other. As P2P systems became larger and more complex, more research has focused on this aspect. Several approaches have resulted, including unstructured networks, clustering, distributed hash tables (DHTs), and super-peer (SP) networks.

Unstructured networks impose no constraints on the placement of data or peers, and are the simplest topology. Querying is done by recursively sending the query to all neighbours, until the data is found or a time-to-live threshold exceeded (flooding). Clustering is the process of grouping peers together according to some constraints, usually peers' interests. Distributed Hash Tables (DHTs) require peers to store indexes of other peers' resources. These can then be used to direct queries to peers that contain the relevant resources. Super-peer (SP) networks treat peers differently based on their resources and network behaviour. Peers that are connected for longer and have more resources are classified as super-peers (SPs). All other peers are known as normal peers (NPs). SPs can be used in a variety of ways to create efficient topologies, such as HyperCuP [22]. SP networks have shown good performance and have consequently become very popular. Yet, they can be very inefficient if not implemented properly. They should be built incrementally and should be able to adapt to the environment. Several questions need to be addressed when creating SP topologies, including how SPs connect to each other and what is a good ratio of NPs to SPs.

2.2 P2P Systems

Napster [28] was the first popular file-sharing P2P system. The system consists of several central index servers that act as directories. A server stores metadata of all peers connected to it, including IP addresses and shared filenames. Peers register with one of these servers when they join the network. They are then able to use it to search for files located on other peers currently connected to the network by sending queries to the server. A notable disadvantage with the Napster design is its simple topology. The central servers form a single point of failure. If a server goes down then all the peers connected to it go down as well. These peers could connect to another server, but this would cause scalability problems, as the server's workload would increase considerably. Napster has two advantages in that it offers fast query processing and fast updating of available resources, but its search is limited to keyword lookups.

Gnutella [30] is a completely decentralised system where peers connect to neighbouring peers to form a massive collection of interconnected nodes. . Due to its decentralised design and lack of central control, it is easy to create ad-hoc

networks using Gnutella. Querying is in the form of keyword lookups and is done by recursively passing on a query (flooding). This unfortunately consumes an unnecessarily large amount of resources. The messages passed in the network use a time-to-live counter. If the time-to-live limit is set too high then a message may loop in the network; if too low then the message may only return a subset of the potential results. Apart from the severe scalability problems that Gnutella faces, there are also problems of inefficiency and denial of service attacks [17]. The protocol used for answering queries is expensive but useful in an environment where there is a lot of peer activity, i.e. not only do peers constantly change, but so do the resources offered by those peers.

Chord is a distributed lookup protocol that addresses the problem of efficiently locating a peer that stores a particular data item [27]. It provides support for just one operation: mapping a key to a peer. Data location can be easily implemented on top of Chord by associating a key with each data item. Therefore, each peer stores a hash of the resources offered by it. This hashmap is then stored by at least one neighbour to quickly search for a particular resource. Chord adapts efficiently as peers join and leave the system, and can answer queries even if the system is continuously changing. Unfortunately, Chord can only locate an item if it is given the exact key that maps to that particular item. This limits how the system can perform searches. Users cannot use phrases or keywords that do not exactly match the relevant key.

3 Schema Interoperability

In order for different databases to communicate with each other, semantic mappings between the databases' schemas are needed. A semantic mapping is a link between logically equivalent attribute(s) in different schemas. For example, a semantic mapping could exist between the attribute *FName* in schema *A* and the attribute *first_name* in schema *B*. Defining these mappings is difficult as database schemas are typically designed independently and, consequently, often differ in many regards. These differences include schemas that use different structures, naming conventions and data models. Furthermore, the same attribute name may have different semantics in different schemas, formulae may need to be applied to attributes in one schema to match attributes in another schema, and one attribute may correspond to several attributes in another schema. All these differences pose significant challenges when matching two different schemas.

These mappings can be determined manually by domain experts or semi-automatically using schema matching. Domain experts are useful as they can define accurate and extensive mappings. However, the process is very time consuming, tedious and expensive. Schema matching is explained in more detail next as it has been used in Mapster to achieve schema interoperability. The various problems and challenges associated with schema matching are covered in more detail in [23].

3.1 Schema Matching

Schema matching is the process of computing the semantic mappings between two schemas that pass user validation [18]. Due to the complex nature of potential matches, fully automatic matching is deemed infeasible [18]. For this reason, user interaction during a semi-automated match process, and user validation after it, is recommended. Besides the two schemas, input can also include auxiliary information, like dictionaries.

Several schema matching techniques have been developed over the years. An individual technique is called a matcher and can be applied by itself or in combination with others. These matchers work at one of three levels: schema structure, attributes and stored values. Structure matchers typically examine the path from the root of the schema to the current attribute. They analyse the full path name, typically using linguistic analysis, and perform subpath matching. Other structure matchers increase the similarity of attributes based on common neighbours. Cupid [18] and Similarity Flooding [19] are two systems that make use of these techniques. Attribute matchers are the most popular and typically analyse the attribute names using dictionaries and thesauri to find matching attributes. Data types and schema constraints are occasionally used. Finally, attributes' stored values can be analysed to find matches, typically using machine-learning techniques e.g. Bayesian Networks. Although useful data can be found in stored values, it does mean that far more processing needs to be done. iMAP [8] and LSD [3] both use machine learning at this level.

Matchers are usually combined to improve match predictions. Combination can be achieved by either using the hybrid

approach or the composite approach. The hybrid approach is more common but less flexible, as the selection of matchers and their execution order are fixed. This approach is useful when the problem domain is known quite well and the combination of matchers can be tuned beforehand. With the composite approach, the selection and execution order of matchers can be chosen at runtime. This allows the system to adjust for different environments. COMA [9] seems to be the only system that has taken the latter approach.

3.2 Use of Mappings

There are two common methods of defining and using semantic mappings. Two P2PDB systems, Piazza [12] and Hyperion [1], use pairwise mappings. A pairwise mapping is a mapping from attributes in one database to attributes within another database. These mappings are usually defined manually, which allows them to be accurate, but this is labour-intensive. Countless mappings need to be generated, as a mapping is required for every pair of related attributes. Existing mappings can be used to determine new transitive mappings, making pairwise mappings quite flexible. However, a network of mappings can become exceedingly complex, as sequences of mappings must often be traversed in order to determine how peer A matches peer Z. This makes scalability problematic for pairwise mapping.

An alternative method is to use a mediated schema. This method has been used by the P2PDB system Edutella [21]. A mediated schema is a single schema that is the combination of several schemas. Mappings are then simply links between individual schemas' attributes and the attributes within the mediated schema. Fewer mappings need to be created when a mediated schema is used, as the schemas are only mapped to a single mediated schema. No mappings need to be defined between the different schemas, making the mediated schema approach simpler and more scalable. Query processing is easier as the query only needs to be translated once for each schema, being from the mediated schema to the relevant schema. A disadvantage of using a mediated schema is that it can be difficult to maintain a complete up-to-date version, especially in a dynamic P2P network.

4 Mapster Architecture

A P2PDB system is built upon a P2P platform which allows nodes to communicate over the network and provides basic topology management. Peer communication is important, as numerous messages need to be sent across the network, such as database queries, query results, topology adaptation instructions, and coordination messages. Coordination messages are vital for handling communication errors.

Query processing and a GUI are also necessary components. Queries and their results are transported along the network. This transportation is dictated by the P2P platform and, more specifically, by the topology management scheme. Queries also need to be translated for each peer according to the peer's schema. This is done by the semantic interoperability component, either before queries are sent to peers or when each peer receives the query. Post-processing of results, e.g. for aggregation, is also needed. Although the query GUI can be basic, it is helpful to have a good interface for querying and displaying the results. Other components can be added to a system. These can include security enforcement, advanced GUI features, efficient topologies, and online editing of database schemas.

Semi-automatic schema matching in Mapster aims to make fine-grained querying of heterogeneous P2P databases possible. While effective schema matching techniques exist, they focus on static, small-scale and close-knit environments. They do not scale well to environments like P2P networks. This is due to several problems, including the dynamic nature of the network, where peers can come and go at will. The size of P2P networks and the speed at which queries need to be processed makes this scaling problem important to address. In addition, there is no central authority, which makes it difficult to keep an up-to-date picture of all the current peers' schemas, consequently making it difficult to provide the peers with an accurate view of available information.

Mapster exploits a cluster topology to make schema matching viable in this context. Only a handful of P2PDB systems have been developed [1,12,21,22] and none of them has focused on exploiting the topology to influence schema matching.

Mapster uses a SP (super-peer) network and clustering of peers according to areas of interest or domains. Clustering peers according to their domain helps to break the network up into sections that are more manageable. These domains are

individually more stable than the entire network and, so, provide a better environment for schema matching. Grouping schemas by their domain also allows the schema matching component to exploit any knowledge specific to those domains, such as ontologies, which has been shown to be effective in e.g. [4].

Each domain has an SP in charge of it. This SP performs the bulk of the decision-making regarding the domain. It maintains a mediated schema (MS) for the domain, which it constructs from the schemas of the connected NPs. A mediated schema is used to represent all the schemas within a single domain. If a peer wishes to join the domain, it first communicates with this SP. The SP will add the new peer's schema to the MS, using schema matching techniques.

SPs from different domains can communicate with each other to support global queries, i.e. queries that are not limited to the current domain or to any particular domain. This is useful, for example, if a user wanted to know if there were any houses for sale that fell within his budget and were close to certain types of transport, then he would need to query the *Property* domain and the *Transport* domain.

Figure 1 illustrates a basic network for Mapster. The large circles represent domains. Each domain contains several NPs, represented by small circles, and a single SP, represented by a large square. The thick arrows indicate the connections between domains. There are two mechanisms that can be used to achieve communication between domains, namely pairwise mappings and JXTA adverts; where JXTA[16] is a set of protocols and primitives for building P2P networks. The major components that make up Mapster are covered next.

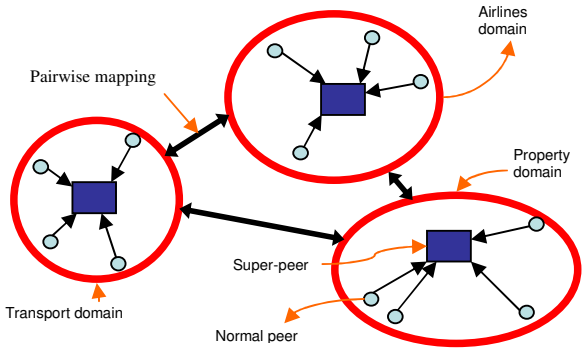

Figure 1: Basic Mapster network

5 Interoperability in Mapster

5.1 Super-Peer Topology

The SP topology localises network changes and reduces the effect these changes have on the surrounding peers. It also pushes peers with low network stability to the edge of the network, where the effects of their dynamic behaviour are minimised. Peers that are not stable can cause disruption to the domain. A peer with low network stability should not become a SP, as the domain will need to be reorganised as soon as the peer goes offline. This reorganisation takes time and should be kept to a minimum.

The previous section explained that a single SP is used to manage a domain. However, this is not very robust and will not scale well when many peers join a domain. To address this scaling issue and the dynamic nature of P2P networks, replication is introduced - having several SPs present within a single domain. These SPs all store the same MS (mediated schema) so that when one SP goes down, the MS is still available. Queries can be distributed amongst the SPs to achieve load balancing. Since the most reliable NPs are made SPs, SPs are unlikely to go offline excessively.

Control in a domain is maintained by having a certain SP act as the root of the domain. This SP performs the bulk of the decision-making. When NPs first connect to a domain, this is the first peer with which they communicate. It is responsible for maintaining the MS and distributing it amongst the remaining SPs. This is the only SP that is visible from outside the domain and is called the virtual SP (VSP) for this reason. The remaining SPs present within the domain are called actual SPs (ASPs).

Some examples of how the topology works are presented next. Figure 2 demonstrates the topology without SP replication, whilst Figure 3 illustrates the topology with SP replication. Figure 2.a shows the setup just after a peer created a new domain. Since the peer started the domain, it also becomes the SP for it. The MS is created using the peer's database. Even though a peer is an SP, it also has the capabilities of an NP. Therefore, a peer which is an SP is represented as both an NP and as an SP in the figure. In the implementation, an SP inherits from the NP class and extends this with SP

capabilities.

Figure 2.b shows what the topology looks like after more peers have joined the domain. If an NP goes offline, the NP's database is flagged as offline in the MS. Flagged entries in the MS are removed if the peer has not reconnected for longer than the Reconnection Period threshhold.

If the SP goes offline, as shown in Figure 2.c by the dotted boxes, then a new SP must be chosen and the MS reconstructed. The MS needs to be reconstructed as the NPs do not store a copy of it. The resulting topology is shown in Figure 2.d. The process of choosing a new SP from candidate NPs is explained in section 7.2.

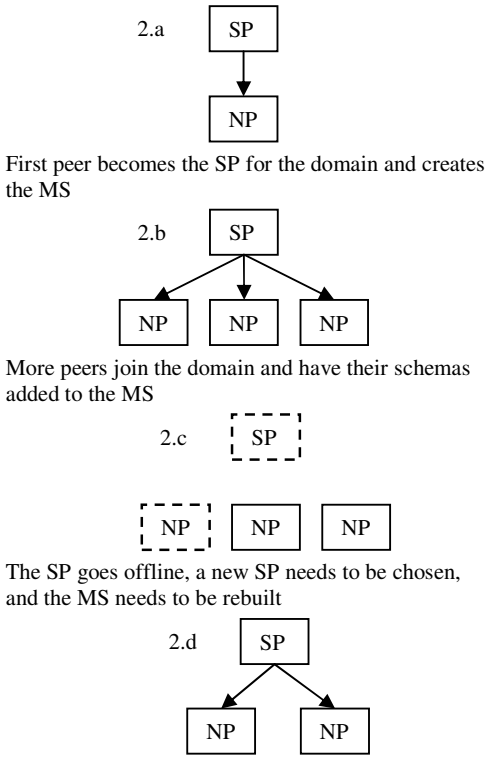

Figure 2: SP topology without SP replication

Figure 3 illustrates how the addition of ASPs increases the robustness of the network. In Figure 3.a a new domain is created by a peer, and in Figure 3.b more peers have joined the domain, requiring a new ASP. Two new scenarios occur in this new topology; namely, when the VSP is changed and when an ASP is changed. If the VSP goes offline, as in Figure 3.c, a new VSP is chosen from the existing ASPs. The new VSP uses its current MS as the new domain MS. SPs can afford to store the MS as they have more resources available to them. This is far better than having to reconstruct the MS, since applying schema matching to all the peers' databases would be time consuming. If an ASP goes offline, then its children NPs ask the VSP to add them to another ASP. This is also shown in Figure 3.c, where the second NP from the first ASP connects to the remaining ASP (figure 3.d).

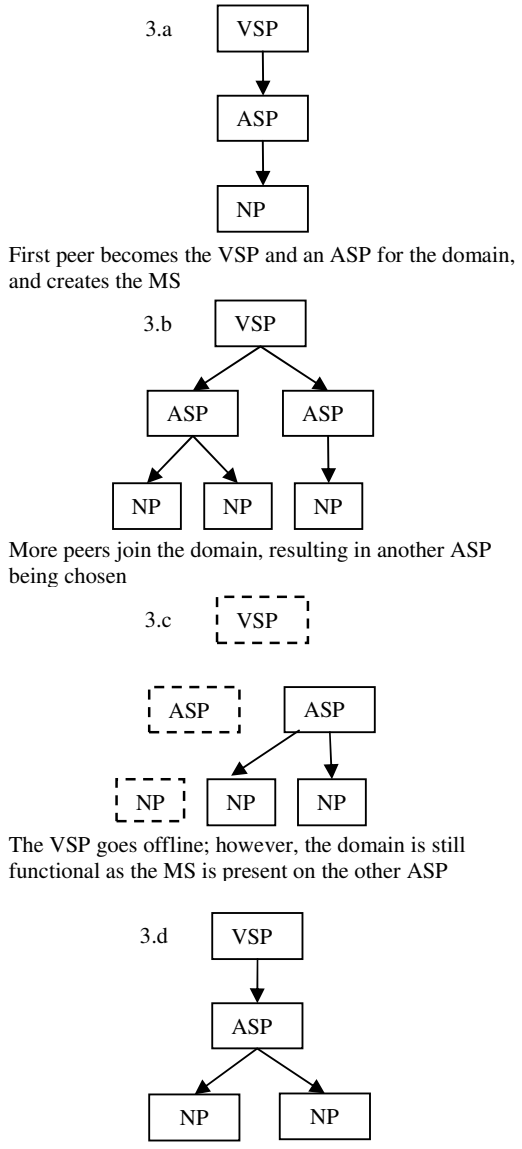

Figure 3: SP topology with SP replication

5.2 Mediated Schemas

Since schema matching can be expensive in dynamic environments, the network is broken up into more stable sections using domains. Mapster assigns peers to domains so that a domain comprises many semantically similar databases. In order to create a MS, schema matching is required. Query processing in systems that use pairwise mappings can require multiple mappings to translate from peer *A* to peer *Z*, whereas an MS approach allows any query to be answered in only two steps (*A* → SP → *Z*), or three steps if peer *A* and peer *Z* are in different domains (*A* → SP → SP → *Z*). This is the worst-case scenario. Hence, the MS approach scales better, which is why it has been used within each domain of the network.

An example of how the MS is constructed is now given. Table 1 to 3 illustrates three simple, slightly different schemas. The MS is created using schema *A*. Since the MS is empty, each attribute from schema *A* is put into its own cluster. Clusters store schema attributes that are semantically similar to each other. A cluster in the MS is analogous to an attribute in an ordinary database. The entries in the cluster sets are in the form *schema.attribute* (the table name has been omitted for brevity). In step 2, schema *B* is added to the MS. Schema matching is used to decide to which cluster each new attribute belongs. After schema *C* has been added in step 3, the MS represents the combination of the three schemas, with each cluster comprising semantically similar attributes. For instance, *Age* was not put into cluster one even though its instances are similar to those of cluster one.

Table 1: Schema A

ID	FName	LName
21	Gareth	Louw
22	Ben	Smith
23	John	Black

Step 1: MS is created using schema A
Cluster 1 = {A.ID}
Cluster 2 = {A.FName}
Cluster 3 = {A.LName}

Table 2: Schema B

Index	First_name
99	Michelle
100	Mike
101	Mike
102	Steve

Step 2: Schema B is added
Cluster 1 = {A.ID, B.Index}
Cluster 2 = {A.FName, B.First_name}
Cluster 3 = {A.LName}

Table 3: Schema C

Index	Age	FirstName	Surname
1	31	Lisa	Blake
2	25	Ben	Smith
3	56	Peter	Black

Step 3: Schema C is added
Cluster 1 = {A.ID, B.Index, C.Index}
Cluster 2 = {A.FName, B.First_name, C.FirstName}
Cluster 3 = {A.LName, C.Surname}
Cluster 4 = {C.Age}

5.3 Schema Matching

Schema matching is used in the construction of the MS. Mappings from the attributes of a new database to the clusters in the MS are found semi-automatically using schema matching techniques. These techniques allow the MS to be constructed far more quickly and effortlessly than manually creating them. Once the mappings have been defined, they are stored with the MS and used by the query processor.

Schema matching is a complex task, so a Mapster utility is provided that enables the automated part of schema matching to be tuned to a particular domain. Specific matchers can be selected and their parameters and combination method adjusted. Results can be evaluated against a set of manually defined matches, which are considered[1] to represent the perfect mappings between the two input schemas. Four evaluation measures are calculated, viz. *recall, precision, F-measure* and *overall* [10]. The figure below illustrates the overall matching process. Section 7.4 covers the match process in more detail, while section 8.1 explains the evaluation measures and how they are used.

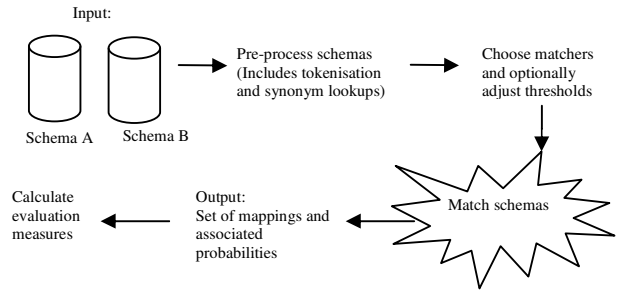

Figure 4: Schema matching process

6 Query Processing

Query processing is the last step in achieving a P2PDB

[1] Schema matching is subjective and, so, perfect mappings may differ from user to user

system. Once a peer has joined a domain and had its database added to the MS, it will then want to make use of the resources available to it. These resources are the other peers' databases and there are two ways of accessing them: browsing or querying. Browsing forces the user to explore all the peers' databases in order to find something he wants. Querying is far more powerful and efficient, as it allows the user to specify, in SQL, which attributes he wants to view, what values they should have and how they should relate to other attributes, either in the same database or in other databases. Browsing is not supported for efficiency reasons. Queries are specified in terms of the peer's own database. If this is not adequate then the query can be specified in terms of the MS. Queries written in terms of the local schema are preferred as the user should understand his database better than the MS and, consequently, be able to specify a query more accurately. Queries on the MS allow attributes not present in the user's own database to be included when querying the network.

Once a query has been specified, it is sent to the NP's parent ASP. The ASP uses the MS to divide the query into subqueries, based on which NPs contain attributes in the query. These subqueries are then sent to the relevant NPs. Once processed, results are sent directly back to the NP that posed the query. Any post-processing, such as aggregation, is then performed by that NP. If a peer fails to return a result this does not affect the post-processing, which simply consolidates information from those nodes that did respond.

If a query is phrased in terms of the peer's own schema it is first reformulated in terms of the MS. The algorithm first extracts all attributes in the query; where there is a wildcard it is replaced with all the attributes in that relation according to the semantics of SQL. The attributes are stored in a hash map, with the key being the attribute in the query and the object being the corresponding cluster in the mediated schema. For each peer that has data relevant to this query, this hash map is then used to generate the corresponding query for the peer.

For example, the query

Select cust.*, day, fee from cust, sales where age>21

might translate to the following for different peers:
P1: Select name,age,day,fee from cust,sales where age>21
P2: Select fname,age from personnel where age > 21
P4: Select name, age, saleDate from sales where age > 21
if e.g. P2 does not have attributes in the day or fee cluster, and P3 does not store data equivalent to the age attribute.

7 Implementation

The four main Mapster components are a JXTA [16] platform, a topology manager, a mediated schema constructor, and a schema matching utility. These components are outlined and then their interaction in our P2PDB architecture is presented.

7.1 JXTA Database Sharing

JXTA is an open-source P2P library, which allows a variety of devices, ranging from cell phones to high-end servers, to form a P2P network. It aims to work independently of the programming language, operating system, and network protocol used by any peer [11]. JXTA was used to create the underlying P2P system, as it provided all the basic P2P functionality. The primary form of communication in JXTA are adverts, which describe a resource in XML.

A custom JXTA advert type was created to describe a database, which could be used to advertise peers' databases on the network. Whenever a new peer connects to the JXTA network, a database advert is created for the peer's database and stored in the new peer's local cache. This allows another peer to retrieve the advert from that peer's local cache at any time, assuming that it is still valid, as adverts contain a time-to-live value. Once a peer has obtained another peer's database advert, it can use the advert to view and query the peer's schema and data.

Figure 5 illustrates how the JXTA component works. A user can connect to the network, as shown by point 1. The peer can then advertise its database (2) and retrieve other peers' database adverts (3). Once an appropriate advert has been received, other peers can use that advert to request data from that database (4). This request uses information from the advert to send an instruction to the relevant peer to return the corresponding data (5).

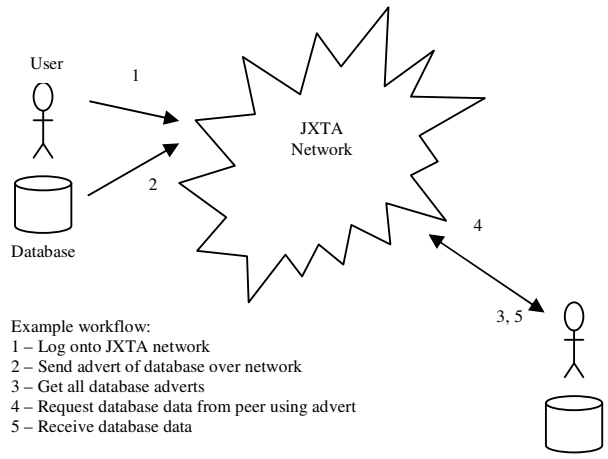

Figure 5: JXTA database sharing outline

7.2 Super-Peer Topology Manager

A topology manager provides a layer above the JXTA network which enables data access to be controlled by domain super-peers. This component also automatically adjusts the network under high loads. When too many NPs try to connect to an SP, a new SP is automatically created from the most willing NP. Willingness to become an SP is calculated using peer information such as average CPU load, network behaviour, bandwidth capabilities, etc. This value is called the SP willingness. It is adjusted over time to reflect the peer's behaviour. The user can also explicitly set a peer's SP willingness value, e.g. if the user would like to have the peer become an SP or definitely not become an SP. Every peer also has an NP acceptance value, which is the maximum number of NPs that the node can manage should it become an SP. New NPs connect to the most accepting SP in a domain, based on its NP acceptance level and the number of NPs already connected to it.

The topology manager aims to build a balanced tree, where the VSP is the root and the NPs are the leaves. However, each SP can adjust its NP acceptance value, and SP willingness values change over time. Thus the topology may not be balanced structurally, but balanced in terms of SP load. The ability to calculate how willing an NP is to become an SP is useful in stabilising a domain. Since only the most willing NPs become SPs, the SPs within a domain should be quite stable. Every time a new peer joins the domain, its willingness to be an SP is calculated. If it exceeds that of some existing SP by more than a threshold amount, then it replaces the least willing SP. This ensures that the domain remains stable in the long term, by sacrificing some stability in the short term, and also ensures that such reorganisation does not occur too frequently.

7.3 Mediated Schema Construction

The MS construction algorithm is incremental, which allows a database to be added to a domain at any point with minimal interference to the working of the system. The algorithm was adapted from the work done in WISE-Integrator [13]. The first database is broken up into clusters, by putting each attribute into its own cluster. These clusters become the initial MS attributes.

When a new database is added to a domain, each attribute *f* within that database is compared to every cluster within the MS of that domain. *f* must be compared to every attribute within the cluster to ensure a high degree of accuracy. Comparison is done using schema matching techniques. *f* is then added to the cluster that has the highest match similarity. However, if the match similarity is below a given threshold for all clusters then *f* is put into a new cluster. This process is performed for every attribute in the new database. Section 5.2 contains an illustrated example of this process. The algorithm presented below outlines the MS construction process.

```
For every attribute f1 in the new database
    For every cluster in the MS
        For every attribute f2 in the current cluster
            Compare f1 and f2 using schema matching
            Add the match probability to the cluster score
    c_max = cluster with the highest match probability
    If the match probability of c_max >= limit  then
        Add f1 to c_max
    Else
        Create a new cluster for f1
    Update the schema mapping table
```

Every time an attribute is added to a cluster in the MS, an entry is made in a schema mapping table. These mappings are used in query processing to redirect queries to all relevant peers and to link an attribute in the MS to the corresponding data at each such peer.

When a peer disconnects, attributes from the relevant database

are flagged as offline. They are not removed from the MS to avoid having the peer go through the schema matching process again. The mappings are removed only if the NP does not reconnect after a specified time, currently 20 days. However, the mappings are still available on the NP itself for any future use.

7.4 Schema Matching

There are currently six matchers: three attribute matchers, two structure matchers and one instance matcher. The most accurate attribute matcher tokenises the attribute name and then uses WordNet [30] to lookup all synonyms of the base form of each token. Once all synonyms have been found, similarity is calculated as follows [28]:

$$name\_similarity = \frac{sum\_of\_common\_synonyms}{(total\_tokens \div 2)}$$

The EditDistance matcher uses the Levenshtein function [29] to calculate the number of linguistic transformations required to turn attribute name *A* into attribute name *B*. This was added as WordNet cannot handle tokens that are not English words. For example, the WordNet matcher will work well for the two attribute names: *FirstName* and *Name.* The EditDistance matcher will work well for the two attribute names: *FName* and *CustName*. The third attribute matcher uses a data-type compatibility table to compare attributes' data types, as in [23]. The table specifies how compatible two data types are. This is most useful in excluding inappropriate matches suggested by other matchers.

The two structure matchers are the NamePath and the Similar Neighbour matcher. The NamePath matcher compares the paths of the current attributes using an attribute matcher because it is more flexible than string comparison. The second matcher is called the Similar Neighbour matcher. This works as follows: if attribute *A* matches attribute *B* with *x* probability then the neighbours of *A* have their match probability to *B*'s neighbours increased by a fraction. This fraction is usually 10% of *x*, but can be adjusted. This is a common approach in schema matching, which captures the fact that logically related attributes are often grouped together. Therefore, when a match is found, this matcher helps indicate that the neighbouring attributes of the current attributes are probably similar. This matcher runs after all other matchers, using their results to decide which neighbours are affected and by how much.

At the instance level, a simple keyword matcher extracts all words from all the stored values of the current attribute. The most frequently occurring words of one attribute are then compared to those of the other attribute. A similarity value is calculated based on the number of common keywords found.

Each matcher computes a similarity value for each match it proposes, which ranges from zero to one. These match predictions need to be combined in order to benefit from the use of multiple matchers. The default combination technique in Mapster is to average the similarity values, which was shown by COMA [9] to outperform all other combination techniques. There are two other combination methods available, namely minimum and maximum, which can be requested instead using the schema matching utility.

7.5 Complete System

The components discussed above are used in Mapster as indicated in figure 6. The sequence of work is shown on the left of the diagram; the text in brackets indicates which component is responsible for that work. Note, in point 3, that an SP periodically updates the domain advert. The advert is not actually sent out over the network, but is simply put into the SP's local cache. This reduces network traffic, as the advert is only transmitted when a peer asks for it. Figure 7 shows the Mapster software components that exist on each node, and how these interact with external entities viz. users and databases at the node, and the JXTA P2P network infrastructure.

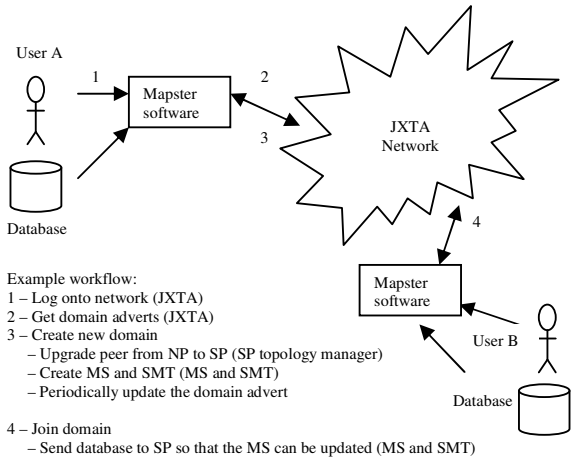

Figure 6: Mapster Example

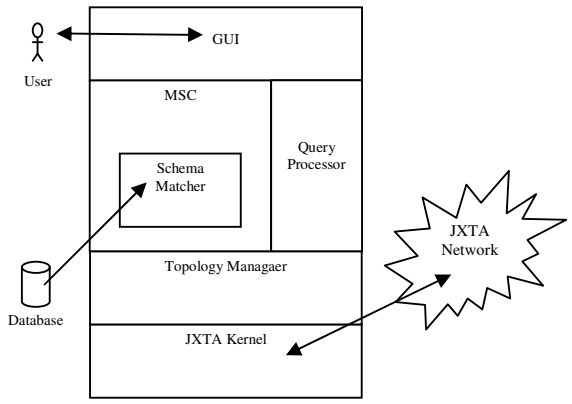

Figure 7: Mapster Software

The first task a peer has is to connect to the network. This requires the user to specify his database and to log onto the network so that JXTA can assign a unique ID to the peer. The peer then needs to create a domain or join an existing one. Let us assume the peer creates a new domain. The creation of a domain is handled by the topology manager. The peer is upgraded to the VSP. Once the VSP is running, the initial MS is built by the schema matcher.

The domain is now up and running. A domain advert is periodically updated and published to notify other peers of its existence and status. This status includes information like the number of peers in the domain, the subject of the domain, and its uptime. The advert and the way in which it is published are handled by the JXTA component.

A new peer can connect to the network and browse domains. The peer can then join a domain by extracting the VSP's address from the domain advert. The peer sends its database to the VSP, where the schema matcher proposes mappings from the peer's database to the domain MS. These are sent to the new peer for user validation. The confirmed/altered mappings are sent back to the VSP, where they are added to the MS and the peer is added to the domain. The VSP decides where to put the new peer, either by making it an ASP or by adding it to the most accepting ASP.

Handling nodes that leave the network

Research into the JXTA library showed that their solution of having up-to-date peer information was only a proposal and not yet implemented. Consequently, a pinging component was added to Mapster. Peers need to ping each other continuously in order to have fresh information about the peers connected to them. A ping is done by trying to open a connection to the peer. The ping operation will timeout after 10 seconds if a connection cannot be opened. Pinging is done according to the topology, i.e. an NP will ping an ASP and an ASP will ping a VSP. If an ASP has not heard from the NP after 45 seconds then it will try to ping it. If this fails then the NP is marked as offline in the mediated schema and is removed from the list of children on the ASP. The same applies to the VSP pinging ASPs. If a peer cannot ping its parent ASP then it will try to ping the VSP. If the VSP is online then the peer will ask the VSP to reconnect it to the domain. If the parent was the VSP, then the peer must cooperate with the other ASPs to choose a new VSP. These checks allow the peers to handle the dynamic nature of the network, where peers come and go at random.

Each pinging component is run in its own thread, to prevent it from interfering with the normal operation of the peer. If the peer is an ASP, then it will run two pinging threads: one to ping the VSP and another to ping the NPs connected to it.

8 Evaluation

Three aspects of the system were tested, being the schema matching, the P2P architecture and the usability of the system. The tests aimed at checking the viability of the Mapster approach to schema matching in P2P environments.

8.1 Schema Matching Utility

This utility allows a user to select and combine various matchers in order to optimise the schema matching component of Mapster. All parameters for each matcher can be set, in order to improve match accuracy and execution time. The combination method can also be adjusted. This level of flexibility was useful in fine-tuning the schema matching component of Mapster. Match candidates were evaluated against a set of manually defined matches, which are considered to represent the perfect mappings between the two input schemas. Four evaluation measures were calculated to measure performance and accuracy, viz. *recall, precision, F-measure* and *overall* [10]. While earlier tests covered databases from more than one domain, in the final evaluation thirty-two databases were used which all contained data about university courses and students, so as to maximise the number

of matches to detect. The databases had been created independently by students in a third year course. Each database typically had between three and four relations, and an average of 14 attributes. The general similarity between database schemas was about 75%.

For brevity, only *overa*ll and *precision* results are given here. *Overall* aims to measure the post-match effort required to remove false matches and add missing ones, whereas *precision* measures the proportion of proposed matches accepted by the user. The path matcher achieved an average *overall* score of 0.22 and an average *precision* of 0.64. The name matcher used WordNet and was the second best matcher, with an average *overall* of 0.29 and an average *precision* of 0.7. The edit distance matcher performed poorly, with an average *overall* of 0.06 and *precision* of 0.54. The datatype matcher proved ineffective unless used in combination with other matchers. It scored an average *overall* of -0.03 and an average *precision* of 0.33. The keyword matcher performed the best, with an average *overall* of 0.34 and average *precision* of 0.73. The combination of matchers was optimised - the weighted sum of three matchers proved best for this domain: 45% for the keyword matcher, 45% for the WordNet matcher and 5% for the name path matcher. The matchers executed relatively quickly, with the combination taking, on average, 44 seconds to execute.

8.2 P2P Architecture

Several aspects of the architecture were evaluated as the network grew in size, including joining times, mediated schema size, query times, and reconnection times. Tests involved networks with up to 17 peers. Joining times were used to measure how much strain the SPs were put under as domain membership/size increased. It also checked how well the use of a mediated schema was working. Joining times were linear for all tests. The MS cluster size increased very slowly as the domain grew, showing that it scales very well to the number of schemas present in a domain. With 17 peers connected, the total number of attributes in the domain was over 220, but the size of the MS was only 26. Queries were split into simple and complex queries, and times for each query type were recorded. Complex queries were slightly faster, which can be attributed to the fact that fewer schema attributes and stored instances need to be transmitted along the network. In general, query times grew linearly as the number of peers increased. Reconnection times, i.e. the time taken by online peers to reconnect to a domain when their parents went offline, were very fast - typically less than 4 seconds, because they did not have to add their schema to the MS.

8.3 Usability

We rounded off Mapster's evaluation by checking user satisfaction in an experiment involving 12 participants, six system administrators and six subjects with average computing ability. All were asked to query a P2PDB; the former also had to configure and run the schema matching utility. Participants were observed using the software and then completed a questionnaire afterwards. While a number of minor modifications were suggested, a substantive problem was that subjects disliked using SQL and would have preferred a query-by-example type of interface. This is left for future work. Six system administrators who evaluated the schema matching utility found the matchers and their results easy to understand and were all able to configure Mapster as required.

9 Related work

In this section we compare Mapster to the four other P2PDB systems currently in existence. Piazza [12] is a P2P data management system that provides semantic mediation between peers using semantic pairwise mappings, which are manually defined between pairs of peers. These mappings are then used to compute mappings across the network by exploiting transitive relationships. Manual mapping can be tedious work, especially if it needs to be done for several different peers' schemas. The work done by the Piazza team closely matches the work done with Mapster. However, Mapster attempts to define as many mappings as it can automatically. Mapster is also easier to query as transitive mappings need not be computed.

Edutella [18] is an open source project that has been built on top of JXTA. It uses RDF to provide a metadata infrastructure for P2P applications. RDF was chosen as it is semantically rich and supports extensive querying capabilities. However, RDF is complicated to use. Wrappers need to be applied to all schemas to transform them into schemas represented in the common data model used by the Edutella network. Instead of using wrappers, Mapster uses schema matching. Although

Mapster uses mediated schemas, which can also be perceived as a common data model, it does not force the users to define mappings from their schema to the common data model nor does it require wrappers.

Hyperion is a conventional DBMS augmented with a P2P interoperability layer. The research thus far has focused on the specification and management of the logical metadata that enables data sharing and coordination between the peers [1]. A combination of mapping tables, expressions, and functions are used to achieve data integration between peers. These mapping are typically created manually by domain specialists. Mapping expressions are based on the work proposed in [4], particularly the Local Relational Model (LRM). This model enables general queries to be translated into local queries and new mappings to be found using existing ones. Hyperion uses mechanisms called event-condition-action rules to enforce mapping constraints, including mapping expressions. These mechanisms are analogous to triggers in traditional database systems. The system handles the reconciliation and integration of data at query time, which means that results from queries will reflect the current status of the network.

BestPeer enables peers' databases to be shared to allow users access to more fine-grained information [22]. The topology is self-configurable and clusters peers together over time according to the following theory: peers that answer queries the most often or accurately will usually continue to do so, and hence should be clustered together with the peers that query them. To achieve schema matching, metadata for each schema attribute is provided manually in the form of keyword(s). These keywords are then compared to other attributes' keywords to find matching schema attributes semi-automatically. Transitive mappings are also computed. Query processing is a two-phase process. First, agents are sent out to neighbouring peers to find matching relations relevant to the query, using their keyword-based schema matching approach. These matching relations are then sent back to the peer and a query plan is created and executed. The use of agents is distinctive to this system and highlights collaboration between peers in order to achieve a certain goal. Whilst the use of agents in querying processing is unique, the process is two-phase and may take too long to perform, but no performance figures are available. The schema matching is limited to one-to-one matches and relies on good usage of keywords.

10 Conclusion

This paper describes a system called Mapster that allows users in a P2P network to share their databases. The research addresses problems of heterogeneity and scalability of database sharing in P2P networks. It is also the only P2PDB system that incorporates semi-automatic schema matching. To provide fine-grained access to users' data, Mapster takes the unique approach of exploiting clustered topologies to make schema matching viable in large-scale, dynamic networks. The system uses a super-peer (SP) topology to break the P2P network up into more stable sections. These sections are based on the peers' areas of interest or domains. These domains contain a mediated schema that is created by the SPs using the schema matching techniques.

The construction of the mediated schema is done as peers join the network, so the impact on query processing is minimal. The use of clustering according to the peers' area of interest ensures that the shared schemas contain overlapping data, which greatly improves the accuracy of match predictors. Other P2P database systems mostly use pairwise mappings, which does not scale well, and none uses a structured network topology to incrementally manage mediated schema construction. Our approach is the first to use mediated schemas within clusters and pairwise mapping across clusters, and to make use of the topology of a network to enable effective schema matching. It would not be appropriate to use pairwise mapping within a domain cluster nor to use a mediated schema across domains, since pairwise mapping is not effective where the number of pairs is large, while mediated schemas are inappropriate where there is a low degree of overlap between schemas.

11 Future work

The query processing component of Mapster could be refined to handle complex queries more effectively. These typically require post-processing of peers' results. It is unclear where to do this post-processing. If the super-peer is not under too much load then it could perform the post-processing, otherwise the peer that issued the query must do so. Having the super-peer perform post-processing is better, because it subdivided the query and would know how to integrate the results.

To improve the schema matching component, a Bayesian network instance matcher should be added. Once this exists and query processing has been refined, the entire system will be re-evaluated to measure how well various schema matching techniques and combinations perform in a P2P network and how the topology affects this performance. Experiments to measure the performance of different topologies are also needed to compare our super-peer clusters against alternatives.

12 Acknowledgements

Colin Rouse was supported by an NRF scholarship during his MSc research, as well as a grant from the National University of Singapore during his visit there.

13 References

[1] Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R. J., and Mylopoulos, J. The Hyperion Project - From Data Integration to Data Coordination. SIGMOD Record, Vol. 32, No. 3, September 2003.

[2] Baset, S., and Schulzrinne, H., An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Department of Computer Science, Columbia University, New York NY, 2004.

[3] Berlin, J., and Motro, A. Database Schema Matching Using Machine Learning with Feature Selection. Technical report, Information and Software Engineering Department, George Mason University, Fairfax, VA, 2001.

[4] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. Data Management for Peer-to-Peer Computing: A Vision. Technical Report, Microsoft Research, Microsoft Corporation, 2002.

[5] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. FreeNet: A Distributed Anonymous Storage and Retrieval System. 2002.

[6] Crespo, C., and Garcia-Molina, H. Routing Indices For Peer-to-Peer Systems. Stanford University, 2002.

[7] Daswani, N., Garcia-Molina, H., and Yang, B. Open Problems in Data Sharing Peer-To-Peer Systems. Technical report, Stanford University, 2002.

[8] Dhamankar, R., Lee, Y., Doan, A., Halevy, A., and Domingos, P. iMAP: Discovering Complex Semantic Matches between Database Schemas. SIGMOD 2004 June 13-18, 2004, Paris, France.

[9] Do, H., and Rahm, E. COMA: A system for flexible combination of schema matching approaches, in Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.

[10] Do, H., Melnik, S., and Rahm, E. Comparison of Schema Matching Evaluations. Technical report, University of Leipzig, Leipzig, Germany, 2002.

[11] Gong, L. Project JXTA: A Technology Overview. Technical report, Sun Microsystems, Inc. Palo Alto, CA, USA, 2002.

[12] Halevy, A., Ives, Z., Mork, P., and Tatarinov, I., Piazza: Data Management Infrastructure for Semantic web Applications, Proc. WWW2003, Budapest, May 2003.

[13] He, H., Meng, W., Yu, C., and Wu, Z. Automatic integration of Web search interfaces with WISE-Integrator, from The VLDB Journal, Springer-Verlag, 2004.

[14] Introduction to P2P networks, P2P networks project (2003). http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/Intro.html, accessed on 20/04/2006.

[15] Ives, Z., Khandelwal, N., Kapur, A., and Cakir, M. ORCHESTRA: Rapid, Collaborative Sharing of Dynamic Data. University of Pennsylvania, 2005.

[16] JXTA project homepage (2005): http://www.jxta.org/

[17] Lv, Q., Ranasamy, S. and Shenker, S., Can Heterogeneity make Gnutella Scalable? Lecture Notes in Computer Science, pp. 94 – 103, Springer-Verlag, 2002.

[18] Madhavan, J., Bernstein, P. A., and Rahm, E. Generic Schema Matching with Cupid, Technical Report, Microsoft Research, Microsoft Corporation, 2001.

[19] Melnik, S., Garcia-Molina, H. and Rahm, E. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching, in Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose CA, 2002.

[20] Napster website, http://www.napster.com, last accessed 07/09/2006.

[21] Nejdl, W., Wolf, B., Qu, B., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palm, M., and Risch, T. Edutella: A P2P Networking Infrastructure Based on RDF, WWW2002, May 7–11, 2002, Honolulu, Hawaii, USA.

[22] Ng, W. S., Ooi, B. C., and Tan, K. L. BestPeer - A Self-Configurable P2P System. Technical report, School of Computing, National University of Singapore, 2001.

[23] Rahm, E., and Berstein, P. A. A Survey of Approaches to Automatic Schema Matching. Springer-Verlag, 2001.

[24] Rouse, C. Schema Matching in a Peer-to-Peer Database System, Technical Report, Computer Science Department, University of Cape Town, 2006.

[25] Schlosser, M., Sintek, M., Decker, S., and Nejdl, W. HyperCuP: Hypercubes, Ontologies and Efficient Search on P2P Networks. Technical report, Computer Science Department, Stanford University, 2002.

[26] Shu, Y., Ooi, B. C., and Tan, K. L. Relational Data Sharing in Peer-based Data Management Systems. SIGMOD Record, Vol. 32, No. 3, September 2003.

[27] Stoica, I.. Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. MIT Laboratory for Computer Science, MIT, 2001.

[28] Sun, X., and Rose, E. Automated Schema Matching Techniques: An Exploratory Study. Technical report, Institute of Information and Mathematical Sciences, Massey University, Auckland, New Zealand, 2003.
[29] White, S. Tame the Beast by Matching Similar Strings (2004). http://www.devarticles.com/c/a/Development-Cycles/Tame-the-Beast-by-Matching-Similar-Strings/, accessed on 22 December 2004.
[30] WordNet: A Lexical Dictionary for the English Language, http://wordnet.princeton.edu, last accessed 07/09/2006.
[31] Yang, B., and Garcia-Molina, H. Comparing Hybrid Peer-to-Peer Systems, in Proceedings of the 27th VLDB Conference, Roma, Italy, 2000.