Schema Matching in a Peer-to-Peer Database System

By

Colin Rouse

A thesis submitted for the degree of

Master of Science in Computer Science

University of Cape Town

February 2006

Acknowledgments

Thanks to Professor Sonia Berman for all her help and support. She kept me on track and her feedback on my thesis was incredibly helpful. I would also like to thank Professor Kian-lee Tan at the School of Computing at the University of Singapore for allowing me to study there as an exchange student from May 2005 to September 2005. Finally, thanks to my family for all their encouragement and support, especially my dad and mom.

Abstract

Peer-to-peer or P2P systems are applications that allow a network of peers to share resources in a scalable and efficient manner. My research is concerned with the use of P2P systems for sharing databases. To allow data mediation between peers' databases, schema mappings need to exist, which are mappings between semantically equivalent attributes in different peers' schemas. Mappings can either be defined manually or found semi-automatically using a technique called schema matching. However, schema matching has not been used much in dynamic environments, such as P2P networks. Therefore, this thesis investigates how to enable effective semi-automated schema matching within a P2P network.

This research uses an emerging P2P topology concept called super-peers to provide stability and scalability. Super-peer topologies characterise peers based on their strength within the network. Super-peers can be used to structure the network to limit network changes. This stability is important in providing an environment that is better suited to semi-automatic schema matching. Peers are clustered into domains, where a mediated schema for each domain is constructed using schema matching. The developed system, called Mapster, focuses on determining the mappings between the peers' schemas semi-automatically rather than having domain experts define them manually. Scalability has been neglected in previous P2P database-sharing systems, and the use of super-peers and semi-automatic schema matching addresses this.

The schema matcher performed well with independently created databases and testing showed that the P2P architecture scales well, at least for the number of peers used. Querying databases in such an environment appears to be an exciting and worthwhile avenue for future work, with the long-term benefits potentially having significant impact on electronic data sharing.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Peer-to-Peer (P2P) networks are collections of devices, typically personal computers, which connect directly to each other in order to share resources, such as files and CPU cycles [61]. Peers control their own resources and choose when they are connected to the network. In pure P2P systems, there is no central authority within the network. P2P systems have received a lot of attention in the last decade. There are many systems in development in the research field, including Edutella [42] and Piazza [29], and in the commercial field, including Napster [61] and Gnutella [61].

Napster was the first to gain widespread popularity, gaining over 25 million users within a few months of being released [61]. This simple system used a basic P2P architecture to provide file-sharing services. Since then, many systems have been developed. These systems have varying levels of complexity and their applications include file sharing [61], instant messaging [4] and collaborative work [6].

People have focused on improving and extending various aspects of P2P networks. Speed and reliability have benefited the most. Search is now more efficient and more accurate, especially for file-sharing systems [5]. Search has also been expanded to support more complex queries and not just simple keyword lookups.

1.1 Motivation

The aim of this research is to merge the success that P2P file-sharing systems have had with the functionality offered by database systems. Databases contain structured data and have good query processing engines. A mechanism that would allow sharing independently controlled databases, and to query other users' databases as if they were extensions of one's own database, would be highly beneficial.

P2P networks offer an easy way of allowing people to access other people's databases. Informal techniques, such as using web forms, can be restrictive or impractical, as they can be tedious to setup and maintain. Data access is often limited to form filling and can take time to perform. Other approaches, such as the client-server approach, have been

used previously. The client-server approach requires that data is stored at the server and manipulated using its schema, so users have no control over data structure or storage.

The somewhat ad hoc nature of the P2P architecture is suited to encouraging people to simply join and query the network whenever they wish. Although there are some disadvantages to this approach, when compared to other approaches such as the client-server architecture, which include potentially slower query processing and loss of query expressiveness[1], there are many advantages, such as ease of use and the freedom to manage databases independently, which outweigh the disadvantages.

1.2 Problem definition

Existing P2P systems that share databases [29, 42, 41, 3] typically require too much effort from the user either to convert their database to a format that is similar to those already on the network or to define how their database maps to the numerous other databases on the network. This can easily discourage a person from using the system.

In order to share databases in a dynamic environment, messages need to be passed between the databases. These messages are used to communicate queries and other items. Since a query may be written using the schema from the originating peer's database, it needs to be translated into something that the receiving peer can understand. This translation process uses semantic mappings to convert these messages. Semantic mappings simply indicate how a schema attribute, or a combination of attributes, in one schema maps to an attribute, or combination thereof, in another schema [37]. Semantic mappings are useful as they allow a user to query other peoples' schemas as if they were simply extensions to the user's own schema.

These mappings can be produced manually or using schema matching. Manually defining these mappings can be very time-consuming, tedious and labour-intensive. Hence, it would be beneficial to use schema matching, which semi-automatically finds the similarities between two databases and represents them as semantic mappings [20]. Semi-automatic schema matching is not new, but has not been widely implemented in P2P networks. There are four P2P database-sharing systems, namely Piazza [29],

[1] This may result from incomplete or incorrect schema integration

Edutella [42], Hyperion [3] and BestPeer [41]. All of their schema matching approaches have been included in the schema matching background chapter. Of these four, only BestPeer uses some form of semi-automated schema matching. However, their approach is to simplify match as much as possible and is more of an addition to the system than a full component. The others require mappings to be defined manually by domain experts or the database users.

Schema matching has been implemented in systems where the environments have been stable and relatively small, when compared to P2P networks. Very little testing [21] has been done on how schema matching techniques scale for large numbers of databases, or how they perform in dynamic environments, like P2P networks where the collection of data sources is constantly changing. In order to develop a P2P database-sharing system, schema matching needs to be implemented in a way that is effective in a large, dynamic environment. After researching recent developments in the P2P field, a concept called super-peers was chosen to address these schema matching issues. A super-peer is a peer that is relatively stable over time in terms of network connectivity and available resources, and is thus given a special role in the system [55]. These super-peers have been used to construct a P2P environment, called Mapster [54], in which schema matching techniques can operate effectively.

This research has several applications. For example, it would be useful for people working in tourism to know about accommodation in an area. To allow this sharing of information, a P2P system could be set up that connected companies and allowed them to share their databases. The network could run over the Internet and, since it is ad hoc, only companies that wanted to participate could do so. They would not have to adjust their databases to conform to any precise format, which makes the system easy to use and appealing. As another example, [9] describes the scenario of a pharmacist accessing patient records for a tourist from a doctor's database in the patient's home country. This example describes how medical personnel could easily share information with others irrespective of where they are based or how their database stores the information.

A P2P system to share databases must be effective for the network environment and easy to use. This thesis focuses on sharing relational databases, but the ideas from the research can be extended to other data models.

The effort required to use the system needs to be minimal [62]. A user will need to confirm mappings that are generated by the schema matching process. This can be simplified using a good but simple graphical user interface. A user will then want to query the network. Since databases are being shared, the query processing should be able to handle complex queries, such as SQL queries, and not just simple keyword lookups. This means that a query processing component will also be needed.

1.3 Scope

This research investigates the sharing of databases in a large, decentralised and dynamic environment. The thesis addresses the problem of handling different schemas simply and effectively in this context. Related topics, such as query optimisation and security issues, are beyond the scope of this work.

1.4 Overview of document

This chapter has defined the problem addressed in this thesis and outlined the solution. In the following two chapters, background information on P2P systems and schema matching techniques is given. Mapster is presented in Chapter 4 and its implementation is described in the following chapter. Experiments that were performed to test the schema matching, the P2P architecture and Mapster usability are then presented. The findings are given in Chapter 7. Finally, the conclusion of the research, together with future work, is given.

Chapter 2: Peer-to-Peer Systems

This chapter covers P2P systems to provide insight into what research has been successful and could be used in the design of this system, and what could be changed or improved. P2P networks are first introduced and some P2P systems are then presented. This is followed by a section on P2P topologies and finally, some evaluation techniques.

2.1 Overview

"Peer-to-Peer (P2P) is a form of distributed computing that can be described as the sharing of computer resources, such as files and CPU cycles, by direct transfer between peers" [61].

A recent survey on P2P systems listed the following as characteristics of P2P systems [2]:

- Peers connect directly with other peers
- Peers are responsible for their own data
- Peers can join and leave the network at will
- Peers can act as both clients and servers
- Peers are autonomous with respect to the control and structuring of the network, i.e. there is no central authority

Ideally, peers should only have local knowledge of available data and schemas [2]. However, indices, routing data, neighbouring peer schemas and other information is often also known, as this is helpful in many situations.

Conventional multidatabase systems are founded on key concepts such as those of a global schema, central administrative authority, data integration and permanent participation of databases [3]. Pure P2P systems do away with these concepts in order to be more distributed and flexible.

2.1.1 Benefits

There are numerous benefits offered by P2P over other systems, such as client-server and 3-tier architectures, including local control of resources, anonymity, and high

availability and fault-tolerance [17, 57]. The decentralised control of P2P networks is responsible for most of these advantages.

2.1.2 Applications

P2P systems have several applications. Examples include file-sharing (e.g. Napster [61]), instant messaging and pervasive devices communicating (e.g. Skype [4]), distributed search and indexing to enable "deep" searches of Internet content (e.g. [57]), and sharing CPU cycles and storage resources to better utilize capital investments (e.g. SETI@Home [56]). Other examples exist, such as truly distributed directory name services (DNS) (e.g. CAN [50]), scalable event notification infrastructures [47], new forms of content distribution and delivery (e.g. Akamai [20]), and collaborative work and play, such as Web-based meetings and interactive gaming [57].

2.2 Challenges

P2P systems face three key challenges [17, 2, 42]:

1. Resource management
2. Search
3. Security and Privacy

Security and privacy are beyond the scope of this research and will not be discussed here. Please refer to [28] for more information.

2.2.1 Resource Management

Resource management is difficult due to two characteristics [5]. Firstly, the autonomous nature of peers allows them to change their data when they want, how they want and how often they want. Secondly, the scale and dynamic nature of the system makes it difficult to construct a complete picture of what resources are available and how to best use them, at any given moment.

P2P systems must be designed in a way that will allow them to be scalable and robust. To handle these issues, they must be able to address some of the considerations mentioned in [41, 42], such as:

- Load balancing
- Network bandwidth

- Fairness
- Information redundancy in the network

This list excludes security and privacy issues.

Solutions include storing indices at various points within the network [43], various types of network topologies [55], caching [9], and minimising inappropriate use of network resources [5, 36].

2.2.2 Search

Search is a fundamental operation in P2P systems and is challenging for numerous reasons [44]. These include the absence of a global schema; resource retrieval, and optimisation thereof, in a heterogeneous and distributed environment; and correctness and completeness of query results in a large, dynamic[2] environment. In pure P2P systems, there is no global schema. This makes it difficult to search the different data sources and formats, and to identify semantically equivalent data items. The large size of P2P networks suggests that query results may take a significant amount of time to be retrieved. The dynamic addition and removal of peers also makes it difficult to return results that correctly reflect the resources available in the network at any specific time. Typically, a user will not examine more than the top ten results of a query [61], so result completeness is not as important as result quality.

Due to the distributed and dynamic nature of these systems, resource retrieval is not simple to do effectively and efficiently. A simple approach is to flood the network with queries in order to obtain results. This allows the system to obtain a complete set of results, but at the expense of generating lots of network traffic. Gnutella [61] uses this approach. Another approach is to limit the query expressiveness. As such, keyword lookups are often supported as they can be processed very quickly and indices can be stored for them at multiple peers. CAN [50] and Chord [58] are examples of such systems. These approaches do not adapt very well to the dynamic nature of peers connecting and disconnecting at random [16]. Alternatively, context-aware queries may be supported. These try to match a query string against the metadata of a resource, but are substantially slower and more complex to support [17]. Edutella [42] has proposed a

[2] Peers may connect and disconnect at will

complex network structure and set of protocols to enable their clients to search resources using metadata such as title, author, date, etc.

Another approach is to focus on the topology of the network and how the peers connect to each other. An effective topology will allow for very efficient query passing [17], [55] and a high degree of completeness to the results. Currently, HyperCuP [55] is being researched by the Edutella project as an effective topology. HyperCuP, shown in Figure 1, is an organised topology that allows messages to travel along the network very efficiently. Peers are classified into two groups, namely super-peers and normal peers. The classification is based on peer strength within the network. HyperCuP is covered in more detail in section 2.3.3.

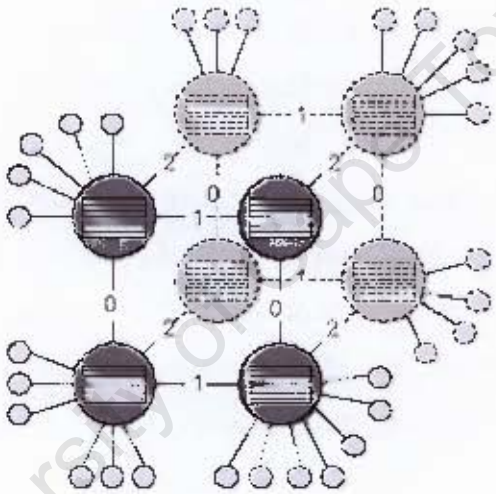


Figure 1: An example of the HyperCuP topology (taken from [55])

One interesting idea, to improve search in random networks, is to take advantage of the power law distribution of naturally occurring networks. Because of preferential attachment to popular peers, networks tend to develop connectivity that has a power law distribution [11]. This law states that a small portion of the peers will have a high connectivity whilst many will have a low connectivity.

Gnutella exhibits a power law distribution [47] and only two modifications are required to implement a power law search algorithm in it [11]. Firstly, instead of broadcasting the query to all neighbours, the query should only be broadcast to the neighbour with the highest number of connections that has not yet received the query. Secondly, peers

need to exchange their stored content with their first- and second-degree neighbours. These changes significantly reduce the number of hops required to answer a query [11]. The most significant reduction is mainly due to the advertising of the node content with their neighbours, because only few nodes need to be visited to examine most of the collection of advertisements and do not need to send the query to every node.

Finally, routing is the technique of deciding what path a message within a network should take. In order for a peer to decide where a message should go, it needs access to information about other peers and their resources. This information is usually stored in a routing index. A routing index is a data structure that, given a query, returns a list of neighbours, ranked according to their approximated ability to answer the query [16]. Essentially, this allows nodes to forward queries to neighbours that are more likely to have answers for those queries. Approximated ability can be defined according to different criteria, such as the number of available resources present at a peer or general query response times of that peer, etc. Several query routing techniques exist, including selective forwarding [55], centralized indexes [61], and relevance driven network crawlers [46]. Two key advantages of routing exist over techniques like distributed hash tables (which use servers to store partitions of a hash-table resource index) [58]. Routing works with any network structure and queries apply to the content of the resources rather than resources identifiers. This makes routing very flexible and a good addition to a modern P2P system.

Several aspects need to be taken into account when using routing for a particular system [5], [16]. Should indexing servers be used and if so then how should they be organised? Should these indexes be replicated at multiple servers? If indexes are used, then what should they store? How should the system deal with users that exhibit dynamic behaviour? For example, users with modems will connect randomly and for short amounts of time, with slow connection speeds. Finally, when should a query be stopped? When all peers have been reached, when a time-to-live counter expires or when enough results have been received?

2.3 Topologies

This section describes P2P topologies and how they affect the performance of P2P systems. An overview of topologies is given, followed by examples of specific types of topologies.

2.3.1 Overview

A network topology defines the arrangement of the nodes in the network [19]. Figure 2 illustrates an example of a P2P network and of a client-server network. Notice how the P2P architecture is more decentralised than that of the client-server architecture.

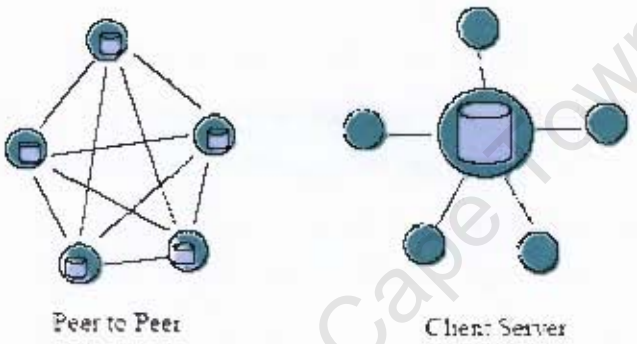


Figure 2: P2P vs. Client-Server (taken from [5])

The topology of a network can affect several factors, including:

- General network bandwidth usage
- Time taken for a message to pass from peer A to peer Z
- Identifying unreachable peers, which affects query completeness
- Reliability[3]
- Scalability
- Load balancing
- Performance under different circumstances, such as very dynamic or relatively static networks
- Message overhead
- Network diameter

[3] Denial of service attacks can be reduced substantially through the use of a good topology

The diameter of a network is the shortest distance between the two most distant peers in terms of node hops. A smaller diameter should reduce information loss and improve query processing time. All the above factors point to one notion: the network topology is fundamental in determining the performance of a P2P system. Several design choices affect the topology of the network, including how peers choose neighbours, if query routing is used and how it is used, content replication, maintenance of peer connections and information, network exploration, and peer role differentiation [5].

There are two types of P2P networks [61]: pure and hybrid. Pure P2P networks treat peers equally, where they may act as both clients and servers, and are completely decentralised. Gnutella is an example of a pure P2P system. Hybrid P2P networks, such as Napster, include centralised components, like servers that store information about peers and handle requests for peer resources. Search can be performed over a centralized directory, while data access can still occur in a P2P fashion. Since centralised servers are used, not all peers are treated equally in hybrid systems. Although pure P2P networks are more decentralised, they do not perform well in practice.

A recent comparison of hybrid P2P file-sharing systems [68] completed at Stanford University investigated four architectures for hybrid P2P servers, namely chained, unchained, full replication and hash. In the chained architecture, servers only store the metadata relating to peers that are connected to them. These servers are linked together, allowing queries to be sequentially sent to different servers. Peer logins are efficient and the servers require the least amount of storage of all the architectures investigated. Unchained networks consist of independent servers that do not communicate with each other, resulting in peers not being able to access all resources in the network. To achieve full replication, each server maintains a complete index of all peer resources, allowing all queries to be answered by any given server. This uses the most storage at a server of any architecture examined. Finally, the hash architecture requires specific subsets of the peers' metadata to be hashed and stored at specific servers. Querying is done by retrieving all these inverted lists and merging them at the local server. The only benefit this technique has over full replication is that it uses less storage at any given server as

the lists are distributed. However, more network traffic is created in transporting these lists to the server processing the query.

A peer login occurs when a peer joins the network and communicates with a server. Peer logins fall into two categories: batch and incremental [68]. A batch login requires the peer to send all its metadata to the nearest server every time it logs into the network. This allows the server to discard metadata of disconnected peers, resulting in more free storage. However, it does create more network traffic. With the incremental approach, only changed metadata is sent to the server by the peer at connection time. This requires query results to be filtered so that resources belonging to disconnected peers are not returned. It was found [68] that the incremental approach outperformed the batch approach. Ultimately, the incremental chained architecture showed the best performance, followed by the incremental unchained architecture.

Other findings from the study included the following. Changing from the unchained to the chained architecture did not affect the performance by much, but did return significantly more results. The batch-unchained architecture is the one that most closely describes Napster's architecture. Batch strategies are far more scalable than the incremental strategies, in terms of storage use. Full replication is the approach to take if network connections are more stable, storage is not an issue, user interests are diverse, and result sets tend to be relatively large. The unchained technique is not recommended in general because it returns relatively few results per query and has only slightly better performance than other architectures. The incremental architecture is useful when systems primarily store historical data, i.e. are "archive-driven". It performs best when sessions are short and network bandwidth is limited.

The above study was done on file-sharing systems and the results are applicable to that type of system. Metadata in file-sharing systems stores information such as filenames and file sizes. This changes as the files that are being shared change. In a database-sharing system, the metadata is unlikely to change as frequently. In file-sharing systems, the number of files and the amount of storage required for file indexes may be large. It is common for systems to share several thousand files at any given time. The target database schemas that will be shared in such P2P systems are likely to be relatively

small when compared to the several thousand files being shared by P2P file-sharing systems. Although the schemas may store thousands of stored values, the schema size itself will typically be small. This is shown in section 3.4, where most schema-matching systems, which are not even P2P systems, do not work with large schemas. Since the schemas are small and do not impose the same restrictions on the system, some of the Stanford findings are not applicable to database-sharing systems.

2.3.2 Unstructured topologies

There are three different approaches to organising unstructured systems [5], namely simple and practical, using past knowledge, and adaptive. The simple approach is the least effective but the easiest to implement and run. All peers are equal and are free to connect to whomever they like. Instead, past knowledge, including usage and peer statistics, can be used to make estimates and guess an effective topology for a given scenario. This approach uses past information that is not updated. Once the topology has been created, it is fixed. This approach may lead to poorly formed topologies and so an adaptive element can be added to form the last approach. An adaptive approach is the hardest to implement, but is the most effective. It works by evolving the network topology over time to best suit the needs of the peers [5].

No global state is kept in unstructured systems. This makes it more difficult to support various services, such as searching, efficiently and effectively. Querying is usually done by flooding the network with the query, as in Gnutella. On the other hand, richer queries can be supported, and, as a result, several P2P systems developed nowadays are unstructured.

2.3.3 Structured topologies

In a structured P2P system, organisation of the network is governed by a globally agreed-upon policy [5]. Data placement and the topology are tightly controlled in such systems. A feature of structured systems is that locating a resource is guaranteed within a bounded number of hops [5]. There are several techniques by which to create structured P2P systems, including clustering, super-peer networks, and distributed hash tables.

Clustering

Clustering is the process of grouping peers together according to constraints. Typically, peers are clustered according to their interests or resources. This technique is effective and is implemented in the majority of the systems in use today [42, 29, 41].

Clustering improves query response time because a query can be sent to a specific collection of peers that are most likely to be able to answer it. It also helps by breaking the network up into smaller segments that are easier to manage. This is useful for several purposes, including updating information and security enforcement. On the other hand, if a cluster cannot be reached[4] then all the resources located within that cluster are temporarily lost. Since queries are sent to the most appropriate clusters and not to all peers, query completeness is not guaranteed.

Super-peer networks

In pure P2P networks, all peers are treated equally. However, peers are typically not equal and this can be used to create inventive topologies. A comprehensive study of the Gnutella network [47] found that a P2P topology becomes important when nodes differ in orders of magnitude of:

- Bandwidth (50 Kbps to 100 Mbps)
- Latency (10 milliseconds to 10 seconds)
- Availability (1% to 99.99%)

These differences show that not all peers are equal and, as a result, should not be treated as such. This introduces the idea of node, or role, differentiation. The study of role differentiation [43] has led to the concept of super-peers. Super-peers are more reliable than normal peers, as they typically:

- Stay connected for longer
- Have more resources
- Remain comparatively constant, in terms of resources and location

[4] This can happen if the cluster is controlled by one or more servers and they go offline

Super-peer topologies can be very inefficient if not implemented properly [43]. They should be applied incrementally and should be able to adapt to the environment. Several questions need to be addressed when creating super-peer topologies, including [55]:

- How should super-peers be connected?
- What is a good ratio of peers to super-peers?
- Is such a ratio needed?

Super-peers can be used in a variety of ways to create topologies. One such topology is called HyperCuP [55] . HyperCuP (see Figure 3) has the following properties:

- Very efficient broadcast and search
- Exploits the topology to reach all nodes in the network with the minimum number of messages possible
- Super-peers or central servers are not essential
- Network is resilient against failure
- Allows for efficient concept-based search

In [55], an efficient HyperCuP construction and maintenance algorithm is provided where nodes can join and leave the self-organizing network at any time and can be made even more efficient using a global ontology of concepts that determine the organization of peers.

For the topology to work, the following requirements need to hold:

- The network diameter should be reasonable
- The number of neighbours must be limited
- Network traffic during search and broadcast should be distributed evenly among nodes in the network
- Hotspots in the network should be avoided
- The system must provide redundancy capabilities

In order to support some of these requirements, each node stores the address of its neighbours. Nodes joining and leaving should not need to transmit many messages. For example, a new node should not have to register with all nodes. Joining hotspots will

end up with more neighbours than other areas and the topology will become unbalanced.

In Figure 3, super-peers are represented as large circles and normal peers are the small circles that connect to the super-peers. The numbers shown on the connections between peers shows how the broadcast algorithm works. The node invoking a broadcast sends the broadcast message to all its neighbours, tagging it with the edge label on which the message was sent. Nodes receiving the message restrict the forwarding of the message to those links tagged with higher edge labels. In the case of an incomplete hypercube, existing peers are temporarily upgraded to super-peers to fill in the gaps. This guarantees that any node will receive a message exactly once and that the last nodes are reached after $\log_b N$ forwarding steps, where b is the branching factor of the super-peers, i.e. the number of connections super-peers can have, and N is the total number of nodes in the network. Link failures, like a peer being disconnected, are handled by the closet neighbour of that node, which executes the departure routine on behalf of that node.

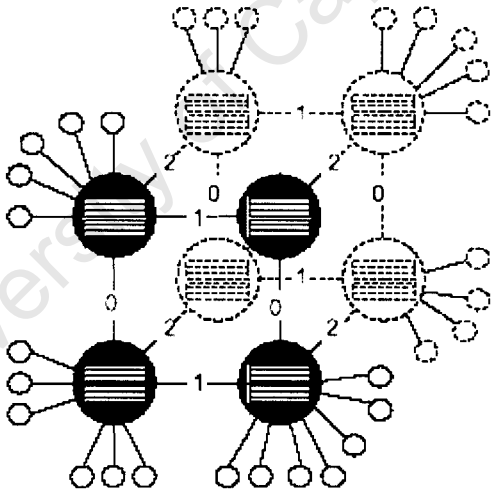


Figure 3: An example of the HyperCuP topology (taken from [55])

Peers can be grouped according to concepts, which can lead to a global ontology.
The network could then be broken up, whereby each concept is a single hypercube. This allows each single hypercube to restructure itself without interfering with other peers in the network. Furthermore, peers may belong to more than one hypercube or concept.

Distributed hash tables (DHTs)

With distributed hash tables, peers search servers, which contain hash-table partitions of the resources offered. Search involves sending a hash of the desired resource to each server. The server with the same hash then asks the relevant peer for the resource and forwards it onto the original peer. Search can be very efficient in these types of systems, but usually at the loss of search expressiveness. This is because the data structures used to make the network efficient only support keyword lookups. CAN [50] and Chord [58] are examples of P2P systems that have implemented distributed hash tables.

2.4 P2P Systems

This section describes three well-known P2P file-sharing systems, followed by four P2P database-sharing systems.

2.4.1 Napster

Napster [61] 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 direct connection is then established between the peer and a peer containing the resource using the IP address of the peer that is returned by the server [61].

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.

2.4.2 Chord

Chord is a distributed lookup protocol that addresses the problem of efficiently locating a peer that stores a particular data item [58]. 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, and storing the key and data item pair at the peer

to which the key maps. Therefore, each peer stores a hash of the resources offered by it, which can be used by other peers 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. Results from theoretical analysis, simulations and experiments show that Chord is scalable, with communication cost and the state maintained by each peer scaling logarithmically with the number of Chord peers [58].

Chord is very efficient at locating a particular data item. Unfortunately, it can only locate that 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.

2.4.3 Gnutella

Gnutella [61] is a completely decentralised system where peers connect to neighbouring peers to form a massive collection of interconnected nodes. Querying is in the form of keyword lookups and is done by recursively passing a query to all neighbours. This effectively sends the query to all peers present within the network. This unfortunately floods the network with messages and consumes an unnecessarily large amount of resources. This inefficiency is because the protocol used in the system was not designed to scale to the number of users that Gnutella now experiences.

Apart from the severe scalability problems that Gnutella faces, there are also problems of inefficiency and denial of service attacks [61]. The messages passed in the network use a time-to-live counter. If this counter limit is set too high then the message may loop in the network. If the counter limit is set too low then the message may not reach all the peers and may only return a subset of the full potential results.

Research [52, 1] has shown that approximately 70% of the peers are exclusively consumers, whilst nearly 50% of all responses are returned by the top 1% of sharing hosts. This indicates that although Gnutella has a decentralised design it mimics a design remarkably similar to that of Napster. The 1% of peers that contribute 50% of the resources essentially acts as central servers and, consequently, Gnutella suffers from the same disadvantages that Napster suffers from.

However, Gnutella does have some advantages. Due to its decentralised design and lack of central control, it is easy to create ad-hoc networks using Gnutella. 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. The query processing returns a high proportion of possible results. Future work improving the routing protocols and querying techniques is planned.

2.4.4 Piazza

Piazza [29] was started to create a robust infrastructure for the Semantic Web. The idea was to create a P2P database management system (PDBMS) that would handle issues such as data management and be able to offer complex query processing of peers' databases. A PDBMS offers expressive and powerful querying services, but heterogeneous schemas of the various peers need to be mapped to make this level of querying possible.

The Piazza project has developed a language for mediating data between peers. The language supports mappings of simple forms of domain structure and of rich document structure. This language is based on XQuery and can map between peers containing RDF data and peers containing XML data. This schema mediation technique is presented in section 3.5.2. They also provide an algorithm to answer queries and do translations across the full range of data, from RDF and its associated ontologies to XML, which has a substantially less expressive schema language.

Peers can contribute a combination of the following resources [31]:

- Data, e.g. XML or RDF instances
- Schemas, e.g. XML schema or OWL ontologies
- Pairwise mappings between peers' schemas
- Computation, i.e. CPU cycles
- Computed data, i.e. cached answers to previous queries

Piazza consists of two main components, namely the query reformulation engine and the query evaluation engine [9]. The query reformulation engine takes a query posed in the preferred schema of the querying peer, and it uses a rewriting algorithm to chain

through the semantic mappings of the peers to output a set of queries over the relevant peers. The query reformulation algorithm is responsible for ensuring that the results are semantically correct. The query evaluation engine produces results as data is returned from the peers on the network. It makes use of the transitive closure of mappings to return all relevant data in the preferred schema of the original peer.

A key aspect of the system is that there may be many alternate mapping paths between any two peers. An important problem is identifying how to prioritize the paths that preserve the most information, while avoiding paths that involve an unnecessary number of peers and resources [30].

Little focus has been put on the topology of the system, until recently. Clustering has been adopted to allow peers to form structures called spheres of cooperation [9]. These spheres will allow for query optimisations that exploit commonalities and available data. Some of the possible optimization criteria include eliminating redundant mappings, reducing the diameter of the PDMS so that information loss in query reformulation will be reduced, and identifying semantically unreachable peers [29].

2.4.5 Edutella

Edutella [42] is a schema-based system, which uses RDF [51] as its foundation schema language [43]. Several services are provided in Edutella, including querying, metadata replication, mapping between peers' resources, mediation between peers' resources and annotation of peers' resources [42].

Data replication, or caching, of the network's mediated schema is useful as it improves the reliability of the network and helps in load balancing. Edutella uses RDF to map the peers' heterogeneous schemas to Edutella's mediated schema [42]. One important characteristic of the RDF language is the ability to use distributed annotations for resources [43]. This allows a peer's schema to reference resources located within other peers. A disadvantage of using RDF is that existing schemas cannot be used to describe the resources. New schemas must be defined for everything and this is not straightforward as RDF is quite complex.

The HyperCuP topology [55], which is covered in section 2.3.3, provides efficient query routing. Super-peers store indices relating to peers currently connected to them. More efficient search capabilities are currently being researched to take advantage of the HyperCuP topology.

2.4.6 Hyperion

Hyperion [3] is a conventional database management system augmented with a P2P interoperability layer. The system allows peers to define their own schemas. The specification method and management of the metadata required to coordinate and share the peers' data is then handled by the system. This approach is different from other systems, such as Edutella, where the system provides uniform access to the collection of heterogeneous data, in that Hyperion handles the reconciliation and integration of data at query time. Data mediation makes use of mapping tables, which are defined by domain experts. This approach is explained in section 3.5.3.

Hyperion offers several advantages over other systems. Since peers can define their own schemas, the system requires minimal work from the user before he can use the system. The system handles the reconciliation and integration of data at query time, which means that query results will reflect the current status of the network. As mappings are defined by domain experts, they should be accurate. The last two points are also disadvantages. Since data mediation is done at query time, query processing will take longer. Having domain experts define the mappings is also problematic, as mappings take time to define and can be tedious even for a small number of schemas. Hence, the approach does not scale well, which is important in P2P networks.

The network topology is unstructured in Hyperion. Peers only have knowledge of their neighbours. There is no use of super-peers or any form of servers [3]. The team is looking into creating a directory service to share and advertise data [3]. This will be used to speed up the discovery of new peers and help in handling areas where there is a high degree of change in peers' data. Another research issue is to enable each peer to discover alternative paths between itself and its acquaintances.

2.4.7 BestPeer

BestPeer [41] is a generic P2P platform that was created to support the development of various P2P tools [44]. BestPeer consists of peers and Location-Independent Global Name Lookup (LIGLO) servers. The LIGLO servers are used to identify peers whose IP addresses change frequently [41]. This allows peers to know who and where other peers are at any given time. The servers continually ping the peers connected to them in order to maintain the correct status of the relevant peers, which is primarily done to know the online status of peers. The LIGLO servers also handle query routing. All peers that are not LIGLO servers are classified as normal peers.

BestPeer uses a self-configurable neighbourhood maintenance policy [70] to organise the peers. This policy defines a topology that eventually leads to peers clustering themselves according to the following theory: peers that answer queries the most often or most accurately will usually continue to do so and, therefore, should be clustered together with the peers that query them [44]. The project team ran several tests with this clustering strategy enabled and disabled, and compared it with Gnutella and a client-server system. The system with the clustering strategy enabled outperformed all other systems [41].

BestPeer has a three-layered architecture, made up of a P2P layer, an agent layer and an object manager layer. The P2P layer is used to interface with the network, whilst the object manager layer is used to administer the local resources. The agent layer is used by services built on top of BestPeer, such as PeerDB. BestPeer supports some complex services, including information retrieval, database querying among different peers and data mining, with the help of several tools that have been built on top of it [70]. PeerDB allows peers to query databases present on the network. The key challenge in querying multiple databases is to match relations that are semantically equivalent. To achieve this, the team employed an IR approach, which is described in section 3.5.4. This technique is used to match relations during query processing, which is a two-phase process. First, agents are sent out to neighbouring peers to find matching relations relevant to the query. 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 the use of collaboration between peers.

2.5 Summary

This chapter introduced P2P networks and several techniques used within systems to make them effective and efficient. P2P topologies, including clustering, super-peers, and distributed hash tables, were then covered. Several existing file-sharing systems, such as Napster and Gnutella, and database-sharing systems, such as Hyperion and Piazza, were outlined.

Chapter 3: Schema Matching

This chapter introduces schema matching. Existing techniques are presented, followed by a section on evaluating schema matchers. Some recent schema matching systems, including four P2P database-sharing systems, are then covered.

3.1 Overview

Schema matching is the task of determining mappings between the attributes of any two input schemas that passes user validation [38]. Schema matching can be applied to several applications, including data integration, schema integration, data warehousing and e-business or scientific workflow [62]. The first two combine to form database integration or database sharing and are relevant here. Schema integration is the process of combing multiple schemas together to create a single schema useable by all [10]. Schema mediation is the process of handling differences between two database schemas so that they can communicate with each other [29].

Since schemas are typically created independently of each other, various differences often exist between them, such as:

- Different structural and naming conventions
- Different data models
- Different semantics for the same attribute label
- Unit measurement may vary (e.g. Pound vs. Dollar)
- Names may be organised differently (e.g. First name vs. Initials)
- Need to apply a formula to one attribute to get the other (e.g. VAT, sales price and profit)
- One attribute may correspond to several in a different schema, these are termed complex mappings

Schemas may not completely capture the semantics of the data they describe and there may be several plausible mappings between two schemas; making the concept of a single best mapping ill defined. Altogether, this makes schema matching inherently subjective.

Most systems that require schema mediation today have schema matching done manually [49], which is undesirable as it is labour intensive, time-consuming, error-prone and expensive, and requires extensive knowledge of the involved schemas. Databases are rapidly increasing in size, which emphasise these problems and highlights the need for some form of automated or semi-automated schema matching.

3.1.1 Input

The majority of schema matching systems accept XML and relational data sources as input. Besides the input databases, additional input may consist of dictionaries, thesauri, user input, global schemas or ontologies, and a collection of previous mappings between other databases[5]. The majority of systems require some form of domain-specific information, such as synonyms, hypernyms, abbreviations and acronyms.

3.1.2 Output

The output of the schema matching process is the set of calculated match candidates and their associated scores (probabilities of matching). These candidates typically single attributes from each schema. A recent system, called iMAP [18], goes further by finding matches between individual attributes and groups of attributes, called complex matches. Little focus has been placed on finding complex matches, which are quite common in real-world schemas, making up to 40% of matches [23]. For example, *list-price* in schema A may match *price * discount-rate* in schema B or *address* may match the concatenation of *street*, *city* and *state*. These matches can be composed of mathematical, textual and Boolean functions, and often involve joins. Complex matches are more difficult to find because there are an unbounded number of potential matches using an unbounded number of functions, attributes and combinations thereof.

Besides determining various types of match candidates, systems can also produce matches with differing cardinalities. Match cardinality in most systems is restricted to 1:1, which means that an attribute from each schema is present in only one match [21]. Higher match cardinalities, such as 1:n, indicate that there are matches from a given attribute in schema A to several attributes in schema B. For example, *Name* in schema

[5] These mappings may not be related to the two input databases, but can be used to calculate new transitive mappings

A may be matched to *CustName*, as well as *EmployeeName*, in schema B. Systems commonly only return the best match candidate for each schema attribute. iMAP, mentioned above, not only presents the best match candidates, but also the top-*k* match candidates. This presents the user with more match options and thus more choice, but it may place extra burden on the user, as he now has to sort through more potential mappings.

3.1.3 User Effort

Automatic schema matching is considered to be infeasible [49] and, therefore, all systems determine match candidates which a user can then accept, reject or change. Furthermore, the user should be able to specify matches for which the system was unable to find satisfactory match candidates, i.e. new matches.

In order to encourage and improve user interaction, user interfaces should allow a person to specify changes easily and to provide additional input, like known matches or domain knowledge, quickly and effortlessly. User effort is required in three stages of the match process, being pre-match effort, during the match process, and post-match effort [62]. If a system is to involve a user in these stages then it should have a good user interface, which has been a relatively neglected part of many schema matching systems. Clio [67] is a system that focuses on mapping controlled by a user and not by schema matching techniques. The interface allows the user to explore the source and target schema. The user then defines mappings examples, which are used by Clio as the basis for creating and refining schema mappings. Focus is placed on allowing the user to verify and rectify the determined mappings based on the results of the mappings. The source schema and the target schema are shown for reference purposes. The target schema has the results of the mappings super-imposed on it to illustrate the determined mappings. Each mapping is also presented as an example where schema relations and attributes are shown.

Pre-match effort can include preparing dictionaries and thesauri, training of machine learning structures, configuring the various parameters of the match algorithms, setting different threshold and weight values, specification of auxiliary information, such as domain synonyms and constraints, and specifying matches and mismatches.

User interaction is occasionally required to guide the algorithms during the match process. In [62], a system was developed that decided which places would benefit the most from user input. The system asks for user input when there is more than one equally good match or where complex matches may be needed.

The post-match effort is needed to correct and improve the match output. This involves adding matches that were not found and removing or adjusting the incorrect candidates.

3.2 Techniques

Many schema matching techniques exist. A single technique is called a matcher. Figure 4 presents an overview of the different approaches and how they can be classified.

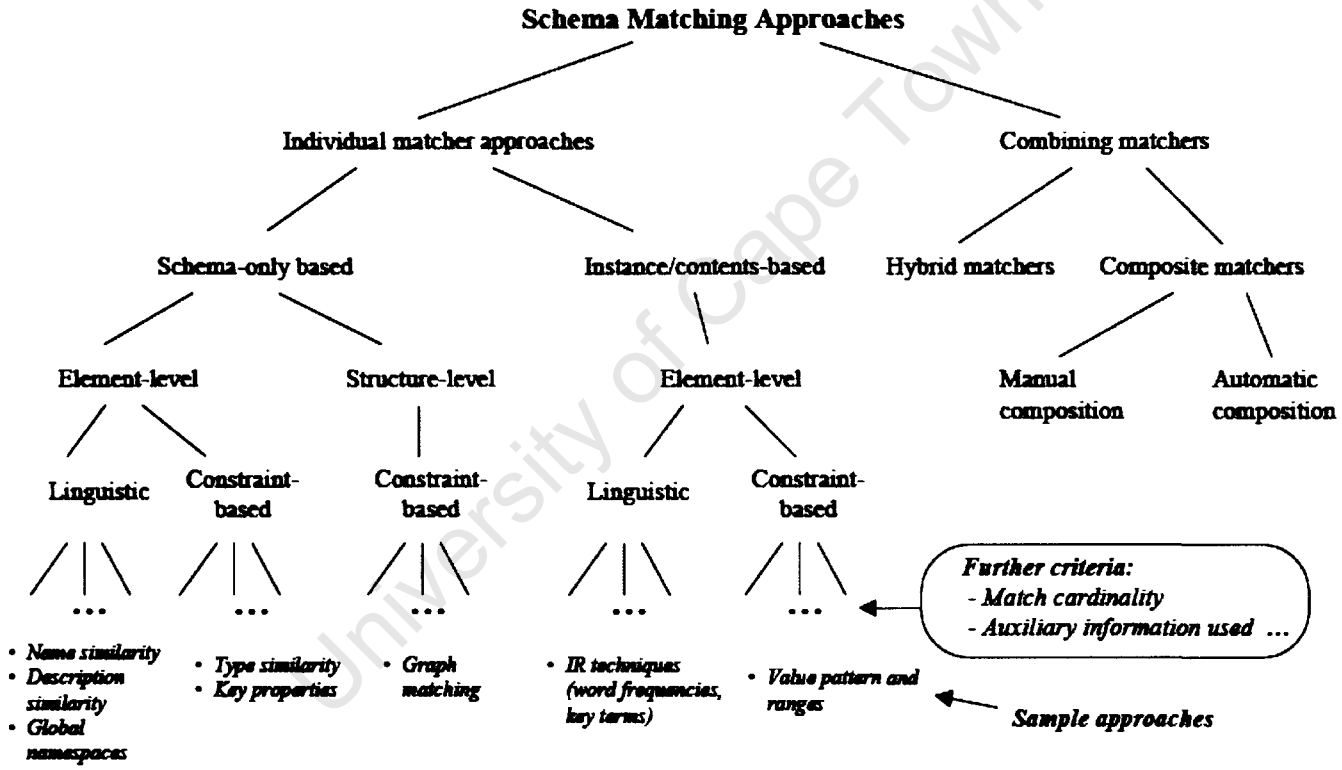


Figure 4: Overview of schema matching approaches (taken from [49])

Matchers can be applied to one of the three levels of the database, being the schema-level, element-level and the instance-level [49]. Some of the more popular techniques that have been used for each level are discussed next. Two general methods can be used for any level. The first is to use past matches between other schemas to determine matches for the two current input schemas. Past matches can be used to calculate new transitive matches. Otherwise, they can be used to train a machine learning technique,

such as a Bayesian Network. MKB is a partial schema matching system that focuses solely on previous matches, which are used to train several instance-level matchers. Little work has been done on reusing previously determined mappings [49]. The use of names from XML namespaces, entire substructures, like people's contact details, schema fragments, or specific dictionaries is already reuse-oriented and should be exploited more so in future systems. The second technique is to use web services as sources of additional information. For example, an acronym web service may be used to help match attribute names or a web service that looks up county names can be used to match stored values, as used in LSD [25].

3.2.1 Schema-Level Matchers

Schema-level matchers essentially analyse the structure of databases. The information that is available from the structure includes general schema structure, integrity constraints, relationship types, such as part-of and is-a, and attribute names [49]. A common assumption used by schema-level matchers, which I refer to as the similar neighbour assumption, is that logically related attributes are often grouped together. This assumption 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 [40]. This fraction is typically 10% of x. Therefore, when a match is found, this assumption helps indicate that the neighbouring attributes of the current schema attributes are probably similar. Neighbours include the attribute's parents, siblings and children.

Referential constraints are used to find joins, which help group logically related attributes together. The combination of finding joins and the similar neighbour assumption make for an effective schema matcher, especially for schemas that are highly decomposed [38, 40, 22, 59, 37]. Referential constraints can be used to create paths from the root of the schema to the attribute. This is useful when the schema is not a tree but a graph, due to shared substructures. These can be contextualised using paths to convert the schema graph into a tree. This allows path matching to occur, for instance, by comparing common subpaths or by using string matching (covered in the next section) to analyse the tokens within the paths. Cupid [38] uses this type of schema matcher to match leaves of a schema tree by comparing the paths, in conjunction with the attributes datatypes. This allows it to determine matches such as the following: the

attributes *City* and *Street*, which belong to *POBillTo,* match *City* and *Street* under *InvoiceTo* in the other schema, rather than the attributes under *DeliverTo*, because *Bill* is a synonym of *Invoice* and not of *Deliver*. Cupid also matches non-leaf attributes if they are linguistically similar and their leaf sets are highly similar, even if their immediate children are not. This is because the leaves represent the atomic data that the schema ultimately describes. This allows the system to handle variations in the structure.

3.2.2 Element-Level Matchers

Element-level matchers look at the attributes present within relations. The information that is available to these matchers includes attribute names, comments and the attribute's datatype [49]. These matchers are the most popular. Almost all systems use element-level matchers and they typically use the attribute name in their analysis, as it provides lots of information in a very short amount of space. The name is usually analysed using different linguistic techniques.

Attribute name matching looks for names that are equal or similar. Equality can be decided using dictionaries and thesauri, and involves equating the fundamental meaning of the word and not just the characters of the words. This involves tokenising the name using punctuation and case. This allows the matcher to deal with special prefix and suffix symbols, such as *CName* and *EmpNO*. The base form of each token is then determined. If a token is an English word then a dictionary is used to find the base form. Synonyms are also retrieved using the dictionary. Most systems [24, 38, 22, 37] use a manually defined dictionary, although one system, SemMa [59], uses a web service called WordNet [65]. WordNet has a very large dictionary, which provides the definition of a word and is able to find synonyms for it. Cupid also removes stop words, including articles and prepositions [38]. This set of tokens and synonyms is compared to the corresponding set of another attribute using the following formula [53]:

$$similarity = \frac{common\_tokens}{total\_tokens \div 2}$$

This learner is good for specific, descriptive names, like *Price* and *House-location*. It does not handle names that do not share synonyms (e.g. *Comment* and *Description*), are partial (e.g. *Office* and *Office-phone*), or lack significant meaning (e.g. *Item* and *Listing*).

Other comparisons can be used that are based on synonyms, for example, *car* can be matched to *automobile*, and hypernyms, for example, *report* is-a *publication* and *article* is-a *publication* implies that *report* might be similar to *article*. Exploiting synonyms and hypernyms requires the use of thesauri or dictionaries. Natural language dictionaries may be useful, perhaps even multi-language dictionaries. In addition, name matching can use domain- or enterprise-specific dictionaries and is-a taxonomies containing common names, synonyms and descriptions of schema attributes, abbreviations, and acronyms. Dictionaries provide valuable matching hints for schemas with relatively flat structures[6].

The similarity of names can also be based on common substrings and edit distances. Edit distance is the number of edit operations required to transform one string to another using the Levenshtein metric [15]. This technique is good at handling non-English words. For example, other matchers would struggle to match *EmpNum* and *EmpNo*, since a dictionary would not be able to use any of the tokens, which are *Emp*, *Num*, and *No*. However, the edit distance can fail in cases where the items are textually similar, but semantically unrelated, such as *Patient* and *Patent*. COMA [22] and Similarity Flooding [40] use this technique.

Comment matching can be very useful. For example, the attribute *ProtSeq* could contain the comment "protein sequence used by LAB Prots". This comment indicates that *ProtSeq* stands for *Protein sequence*, which would have been quite difficult to calculate without the comment [41]. Unfortunately, comments are rarely supplied and, so, are not usually a viable matcher alternative.

Equivalent data types and constraint names, such as *string* being equivalent to *varchar* and *primary key* being similar to *unique*, can be provided by a special synonym table [38]. These techniques are helpful only to limit the number of match candidates and must be combined with other techniques, such as name matching. It is only effective in schemas where the datatypes have been specified very precisely, which is not commonly the case.

[6] As the schema matchers will not be very useful

COMA introduced several instance-level matchers that no other system uses. Common prefixes, suffixes and infixes between the two attribute names were used to calculate the similarities of attributes, using a manually defined thesaurus to compare affixes. The *n*-gram matcher compares strings according to sequences of *n* characters [22]. Different values of *n* lead to variants of this matcher, such as Di-gram (2) and Tri-gram (3). The synonym matcher simply uses relationship-specific similarity values, e.g., 1.0 for a synonym and 0.8 for a hypernym, which represent is-a relationships. The soundex matcher compares the attribute names based on how they sound [22]. For example, *representedBy* can be related to *representative* and *ShipTo* to *Ship2*.

3.2.3 Instance-Level Matchers

Instance-level approaches analyse the stored values of attributes. This level of matching has not been widely implemented because examining every stored value in the input databases can be very expensive. The majority of instance-level matchers use some form of machine learning, which is typically a Naïve Bayes classifier [25]. The Naïve Bayes learner is a very reliable instance-level matcher, especially when the instances describe the attribute name accurately, as *green* and *red* describe the name *Colour*. It does not handle attributes that have short, numeric values, like prices. The Naïve Bayes learner must be trained before it can be used, but it does allow the system to grow and learn over time. Training can be done offline, which saves processing time later on.

Constraint-based characterization, based on stored values, can be applied. This would include numerical value ranges and averages, or character patterns. This technique may allow for the recognition of phone numbers, zip codes, geographical names, addresses, ISBNs, date entries, or money-related entries, based on currency symbols. Several such matchers are implemented in iMAP. A numeric searcher finds matches by examining the value distributions of the various attributes. It supports a limited number of operations, namely addition, subtraction, multiplication and division, between two schema attributes. It does not chain or nest operations together; however, it can use past knowledge to find more complex operations. For example, a past match may define an operation as *VARIABLE * (1 + CONSTANT)*, which could then be used to find matches in the current search space or to find new permutations of operations. A category searcher is used to find matches where the data falls into categories, like gender or age brackets. The categories are found by comparing sets of distinct stored values for a

schema attribute. If the sets are equal or a subset of the sets from the other attribute then they may possibly match. This is useful, for instance, in finding matches between schema attributes that contain weekdays and attributes that just contain *Saturday* and *Sunday*. A schema mismatch searcher matches a stored data value that occurs often in one attribute with an attribute name in the other schema. For example, a schema attribute called *Features* may contain the value "fireplace", whilst in the other schema an attribute may be called *Fireplace*, with values of "yes" and "no". Finally, a date searcher focuses on complex matches that involve date attributes, using a simple ontology. This can be used to infer, for example, that the contents of *BirthDate* maps to the concatenation of stored values *bday*, *bmonth*, and *byear*.

3.2.4 Combination Methods

Individual matchers can be combined to allow for the selection and customisation of matchers based on the application domain and schema types. This allows a system to be fine-tuned for a particular use. The use of multiple matchers also improves the accuracy of the schema matching system [22]. Combination can be done in two ways: hybrid and composite [21]. Hybrid matching is the more common of the two approaches. It uses a fixed combination and execution order of particular matchers, which is not flexible but can be superior for domain-specific applications. It can offer better performance than composite matchers can, as the number of passes over the database can be reduced. Hybrid systems typically use the results from a matcher as input into another matcher.

A composite matcher allows all the matchers to execute separately. In general, matchers do not rely on input from other matchers. Selection of matchers and the combination of their match results can be done either automatically or manually. This provides more flexibility than hybrid matchers do, as a selection of matchers can be chosen at runtime based on the application domain or database schemas, for example. Composite matchers are often used to produce fine-tuned matchers for different domain applications, which are subsequently used as hybrid matchers.

Combining the match scores produced by the various matchers is usually done by calculating the average, weighted sum, or maximum of the scores. Other options are available, such as the stable marriage approach [22], but these still need to be researched. The weighted sum method is the most popular. Systems that use machine

learning [25, 37, 18] tend to use the weighted sum approach, where the weights are learnt as the matchers learn. The final match candidates are often filtered using constraint handlers, which utilize domain and database constraints. An example of a domain constraint would be that a house should only have one selling price. Referential integrity and key constraints are examples of database constraints.

3.3 Evaluation

Evaluation is necessary to find out how effective a system is at addressing a given problem. However, a quantitative comparison of schema matching systems is difficult since matching is an inherently subjective operation [38]. Evaluations done on existing schema matching systems thus far have been lacking in consistency and completeness. Furthermore, systems often focus on specific domains and data models. It has therefore not been possible to compare systems effectively. Future work should focus on creating effective standard evaluation techniques that allows users to choose a system based on various evaluation scores [21].

Ultimately, the goal of schema matching techniques is to reduce the amount of effort required by a user to achieve an accurate set of schema mappings. Up to now, only factors that affect user effort have been tested and not user effort itself [21]. Reliability and completeness have been the main factors tested. There are four popular metrics that are used in schema matching systems, being Precision, Recall, F-Measure and Overall [21]. All of the metrics require a manual match to be done so that results can be compared. The formulae make use of four definitions, being true positives, false positives, true negatives and false negatives. True positives are match candidates that the system predicted correctly. False positives are candidates that were incorrectly proposed. True negatives are candidates that were correctly not proposed. False negatives are candidates that were incorrectly not proposed, i.e. matches that were not identified.

Precision is defined as:

$$\text{Precision} = \frac{(true\_positives)}{(true\_positives) + (false\_positives)}$$

According to [2], this "...reflects the share of real correspondences among all found ones...". This can be interpreted as the proportion of candidates that are correct. The

average value for Precision, for some of the modern systems, is between 0.8 and 0.9 [40, 22, 18].

Recall is defined as:

$$\text{Re}call = \frac{(true\_positives)}{(true\_positives) + (false\_negatives)}$$

According to [2], this "...specifies the share of real correspondences that is found...". This can be interpreted as how complete the match results are. The inverse of Recall, i.e. $\frac{1}{\text{Re}call}$, is the estimate for the effort to add false negatives after the match process has finished. Typical values for Recall lie between 0.8 and 0.9.

Combinations of the above two measurements are used as neither alone can fully access the match quality. F-Measure, which is used in Automatch [8] and COMA [22], and Overall, which is used in COMA [22], Similarity Flooding [40] and iMAP [18], are the two most useful and effective measurements.

F-Measure is much more optimistic than Overall and is defined as:

$$F - Measure = 2 * \frac{(\text{Pr}ecision * \text{Re}call)}{(\text{Pr}ecision + \text{Re}call)}$$

F-Measure scores can range between 0.7 and 0.85 for good systems.

Overall tries to quantify the post-match effort needed for adding false negatives and removing false positives. Overall is recommended for future evaluations by [21]. Overall is defined as:

$$Overall = \text{Re}call * \left(2 - \frac{1}{\text{Pr}ecision}\right)$$

Overall prevents any bias in the construction of the schema matching process to favour recall at the expense of precision and vice-versa. If the score proposed by the Overall measure is below zero, then the post-match effort to correct the results will be higher than the gain from the automatic match process. The most recent systems score an Overall of between 0.45 and 0.8.

3.4 Schema Matching Systems

Schema matching has been around for a while. Only some of the more modern and influential systems are included here, in chronological order. A summary is provided at the end of the section, which highlights the differences between the various systems.

3.4.1 LSD

LSD stands for Learning Source Description [25] and is a composite approach, as shown in Figure 5.

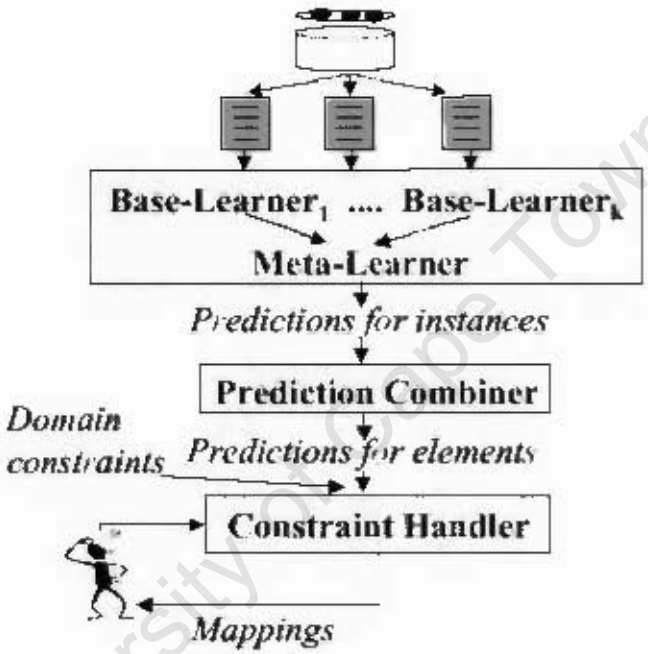


Figure 5: Architecture of LSD (taken from [25])

LSD uses several types of base learners, including an attribute name matcher, a Naïve Bayes learner, and a county-name recognizer, which is an example of a specialised learner that compares stored values against a county-based web database. The determined match candidates are then weighted using the meta-learner and combined using the prediction combiner. 1:1 match cardinality is then enforced with the help of a constraint handler. Each learner is applied to every stored value. The designers argue in [25] that this provides the learners with the best chance of calculating the correct probabilities. However, the system can receive a huge performance penalty.

Experimentation

Four domains were used, each consisting of five, relatively small, data sources, with a manually constructed mediated schema for each domain. Three other sources from that domain were used to train the system. The matching accuracy was evaluated using the Precision measure.

The base learners alone score 42-72%, the meta-learner adds 5-22% and the constraint handler contributes another 7-13%. This results in an overall matching accuracy of 71-92%. The best quality is achieved when all components are combined. The matching accuracy rises quickly and reaches a stable point relatively rapidly. This implies that less data could be used to train the Bayesian network.

Future work will include adding domain-specific recognizers, so that better constraints can be created. These recognizers will also allow the system to make use of constraints earlier and not just at the end, as the constraint handler is currently the bottleneck.

3.4.2 Cupid

Cupid is a hybrid match system that combines a name matcher and datatype matcher with a structural match algorithm [38]. Domain information, including synonyms, hypernyms, abbreviations and acronyms, is used by the name matcher. Keys or object IDs are not processed by the name matcher. Matches are combined by calculating their weighted sum.

Experimentation

Testing [21] was done by comparing Cupid the mappings determined by DIKE [45] and by MOMIS [7]. No quality measures, such as Recall or Precision, were computed.

Cupid was able to identify all necessary mappings for the match task, and thus showed the best results. Using linguistic similarity and structure similarity in combination proved to be far more effective than using either separately. Cupid was able to handle different nesting of schema attributes due to its bias towards leaves in similar neighbours rather than intermediate structure matching. Cupid was able to exploit referential integrity constraints to infer associations from a single relation in one of the schemas to the join of two relations in the other schema. MOMIS and DIKE were

unable to do so. The thesaurus plays a crucial role in linguistic matching [38], but the thesaurus must be manually created and tuning of the control parameters requires expert knowledge of the domain.

3.4.3 Similarity Flooding

Similarity flooding was designed to allow for the quick development of matchers for a broad spectrum of different scenarios [40]. It does not try to outperform custom matchers, but aims to be a useful addition to a more complete system [22]. It is a hybrid system, which uses an edit distance name matcher and a structural similarity flooding algorithm. Unlike most, the system does not use a dictionary and requires very little pre-match effort to perform.

The input schemas are converted into directional acyclic graphs. The similarity flooding algorithm uses the similar neighbour assumption to propagate the similarities, produced by the string matching process, throughout the graph [40]. This is repeated until the similarities of all nodes stabilize. 1:1 match cardinality is enforced using datatype compatibilities and cardinality constraints [40]. Final candidates are chosen using either thresholds or the stable marriage concept [40], which can be used to choose matches that conflict the least with other potential matches.

Experimentation

Limited testing was done using 18 schemas with the number of attributes ranging from 5 to 22 [21]. They were relatively similar, with a 75% similarity on average. Seven users were asked to perform the manual match process. Final match candidates were chosen using the threshold function.

The average Overall measure, over all match tasks and all users, was around 0.6. A quickly converging version of the similarity flooding algorithm did not introduce accuracy penalties and was considered the most useful [40]. The structural algorithm was found to be relatively insensitive to errors in initial similarity values produced by the string matcher.

3.4.4 COMA

COMA is a composite match system used to investigate the effectiveness of different matchers and their combinations [22]. Figure 6 illustrates the COMA architecture.

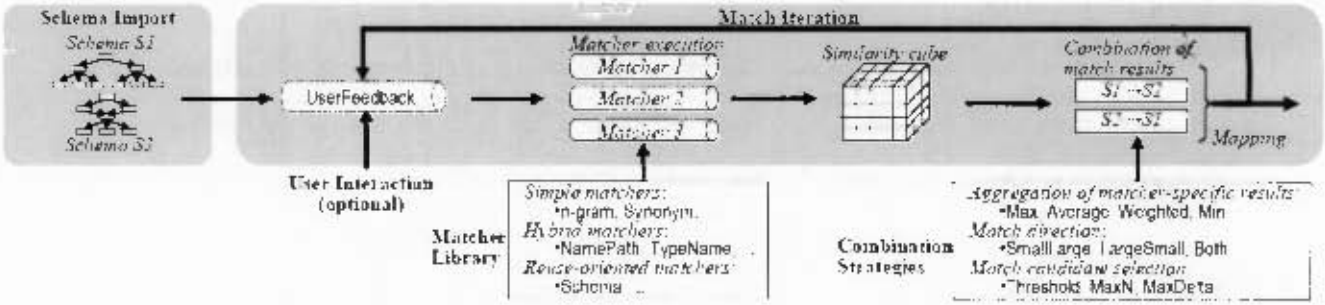


Figure 6: COMA architecture (taken from [22])

The match process can be run either in interactive or in automatic mode. Interactive mode is an iterative process during which the user can specify the match strategy for each iteration, including the matchers to be used, how they are used, match and mismatch relationships, and can adjust the matches made in the previous iteration. Once an effective combination of matchers and other parameters has been selected, this refined match setup can be used in automatic mode for that specific domain. Combination is made up of three steps, as shown in Figure 7:

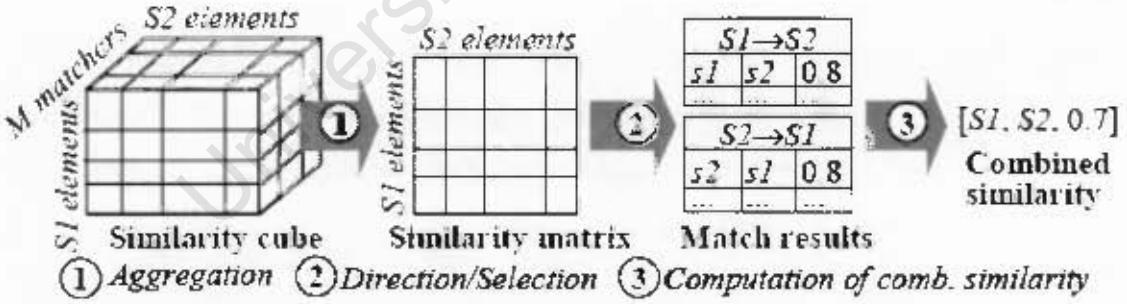


Figure 7: Combination of match results (taken from [22])

Aggregation is done by taking the average of the individual match results. Other available techniques include taking the maximum, minimum or a weighted sum.

Several possibilities are available when performing the selection of match candidates: MaxN, MaxDelta, threshold, and stable marriage. MaxN selects the top N match candidates. MaxDelta selects match candidates that have similarity values that are quite

close to each other. These two options may return match candidates with too little similarity. The threshold option, which selects candidates above the given threshold, may return too many candidates, so a combination of options is used. This is MaxN or MaxDelta in combination with a low threshold, such as 50%. The stable marriage option was left for future work.

The computation of the final combined similarity is done by aggregating all the match results, from the previous step, into one similarity value. This is calculated by either using the average of the results or using a function called Dice [12], which is based on the ratio of attributes that can be matched over the total number of attributes.

COMA supports three different types of matchers, being simple, hybrid and reuse-oriented, as shown in Table 1.

Table 1: Implemented matchers available in the library (taken from [22])

Matcher Type	Matcher	Schema Info	Auxiliary Info
Simple	*Affix*	Element names	-
	n-gram	Element names	-
	Soundex	Element names	-
	EditDistance	Element names	-
	Synonym	Element names	Extern. dictionaries
	DataType	Data types	Data type compatibility table
	UserFeedback	-	User-specified (mis-) matches
Hybrid	*Name*	Element names	-
	NamePath	Names+Paths	-
	TypeName	Data types+Names	-
	Children	Child elements	-
	Leaves	Leaf elements	-
Reuse-oriented	*Schema*	-	Existing schema-level match results

The Children matcher determines the similarity between two non-leaf attributes based on the combined similarity of all their descendants. With the Leaves matcher, only the leaves of the current attribute are involved. This allows the matcher to handle differences in structure.

There are two reuse-oriented matchers. The Schema matcher tries to match entire schemas, whilst the Fragment matcher tries to match collections of attributes, like

Address, i.e. shared substuctures[7]. These matchers can calculate new transitive matches by examining previous mappings.

Experimentation

Extensive testing [22, 21] was done using five XML schemas for purchase orders ranging from 40 to 145 unique attributes. The similarity between the schemas was only around 50%.

In their first set of tests, reuse was not exploited. The option that gave the best overall matches for the aggregation step was average. The most successful combination of possibilities for the selection step were Delta(0.02) and Threshold(0.5) + Delta(0.02). The values in brackets indicate input parameters. For example, 0.5 for the Threshold function means that the threshold was 50%. These combinations scored an Overall of beyond 0.7, whilst Threshold scored the worst. This is interesting as most systems use the threshold option to select the final match candidates. The computation of the combined similarity performed best by computing the average of the results and not using the Dice function.

The reuse of previous matches caused instability in some of the matchers, including Name, TypeName, Children and Leaves. This was because they were not able to distinguish between the different attributes' contexts. Consequently, NamePath, which uses the attribute path, performed better than Name. TypeName also performed better than Name, indicating that incorporating datatype information can be valuable.

Overall, the best no-reuse combination, which included all five hybrid matchers, achieved an average Overall of 0.73. The best reuse combination, which added the Schema matcher to the above selection, reached an average Overall of 0.82. Therefore, the reuse of previous matches is an effective way of improving match accuracy. The best matchers achieved, on average, a Precision of 0.95, Recall of 0.8 and Overall of 0.7.

---

[7] These represent composite attributes, such as *Address*, which is made up *Street Number*, *Street Name*, *Suburb* and *Zip code*.

3.4.5 SemMa

In SemMa, short for Semantic Matcher, structure similarity is computed as follows [59]:

1. If two table names are synonyms, determined by WordNet, and the primary key attributes are synonyms, then these two tables are considered to be similar and the value of *structure_weight* (defined below) is returned
2. However, if at least two attribute pairs in the two tables are individually synonyms, these two tables are considered similar and the value of *structure_weight* is also returned
3. Otherwise, the structure similarity is computed as follows:

$$structure\_similarity = structure\_weight \times \frac{sum\ of\ attribute\_similarities}{total\_number\_of\_attribute\_pairs\_in\_current\_two\_tables}$$

structure_weight is a pre-defined constant that determines the contribution of the structure matcher to the similarity of two elements. *sum_of_attribute_similarities* is the sum of similarities determined by the datatype matcher and name matcher. If *structure_similarity* is above a given threshold then the attributes are considered a match.

Experimentation

Testing involved two medium-sized source schemas and one smaller target schema, each source schema was about 75% similar to the target schema. Recall, Precision and Overall were used to measure the performance of the system.

The combination of all matchers outperformed any other combination of the matchers. SemMa did not perform well if the names of many attributes differed. The use of the schema structure and the dictionary helped in finding more matches but had the side effect of increasing the number of false positives, which lowered the Precision measure. Like previous studies, the results found that the weighting of the datatype matcher should be between 15% and 20% in a hybrid matcher [59]. Future work proposed was building a sophisticated user interface, testing the scalability of the approach, evaluating the performance with larger schemas, and using an ontology to complement WordNet.

3.4.6 MKB

MKB, which stands for Mapping Knowledge Base and is shown in Figure 8, extracts knowledge from a corpus of known schemas and mappings to match new schemas [37]. MKB aims to be a useful addition to a more complete schema matching system.

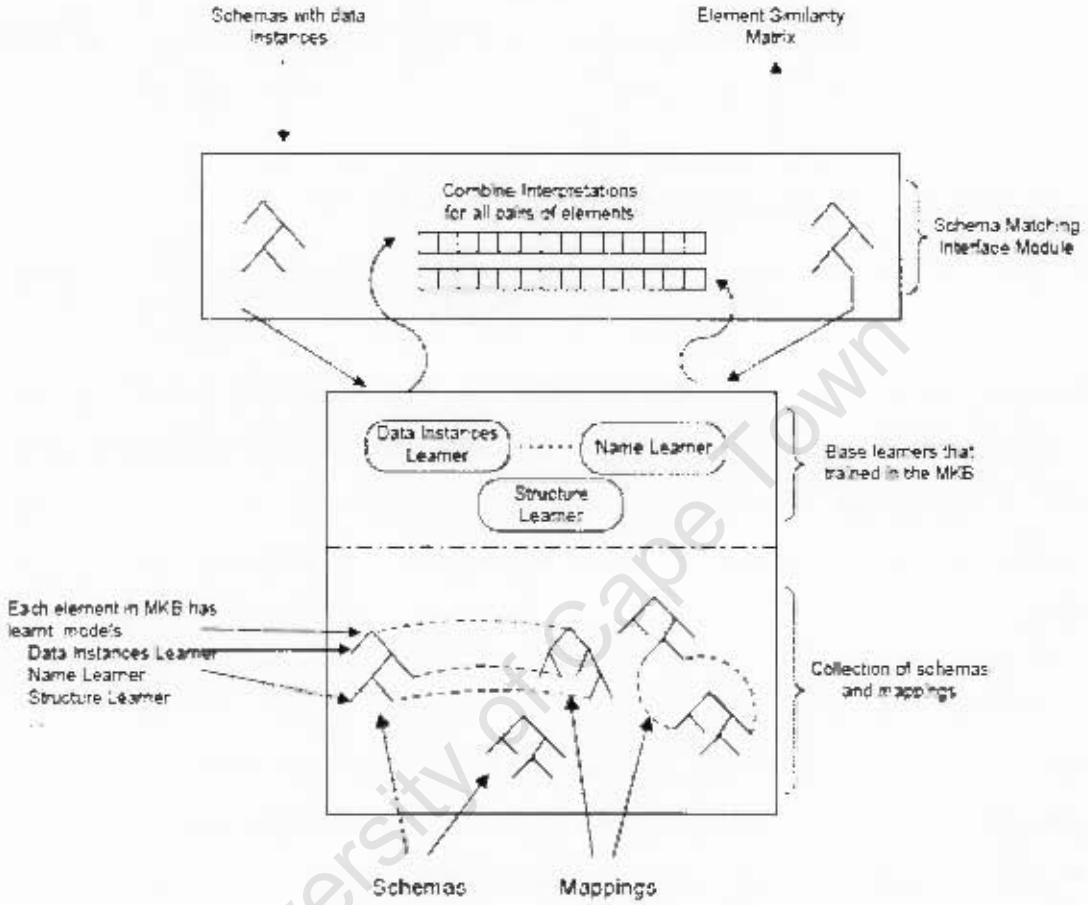


Figure 8: Architecture of the MKB (taken from [37])

In the learned knowledge component are base learners, each are trained to recognize a specific attribute type. Most learners make use of Naïve Bayes classifiers. The instance learner examines features such as the number of words or digits in the stored values, frequency of tokens, and the presence of special symbols, including $, %, and #. A structure learner, which identifies neighbouring attributes that frequently co-occur with an attribute, has also been implemented. Neighbouring attributes are the attribute's parents, siblings and children. It is a useful contribution as neighbourhood patterns being learned for each attribute are specific to that attribute and are unlike generic heuristics that state that attributes are similar if there are other similar attributes in their neighbourhood. A name matcher and an attribute description matcher are also used.

The weighted sum of the match candidates is then calculated. For each attribute in the input schema, the result is a vector representing the calculated similarity that the attribute has to each attribute in the MKB. Attributes from two different schemas can then be compared by calculating a similarity value based on their vectors. This can be the difference of the vectors, the vector dot product, the average weighted difference, or the average significant weighted difference. The average weighted difference weights direct matches between the input schemas more than matches to the MKB. With the average significant weighted difference, only similarities above a given threshold are used in the calculation. The final match candidates are the matches with the highest match similarity.

Experimentation

Several medium-sized relational schemas from the inventory domain, that were created independently, were used for the evaluation. The combination method used was the average weighted difference. The performance of the MKB improved steadily as more mappings were added. The system is more adaptive than LSD and able to learn, whereas other systems, such as COMA, could not [37].

3.4.7 iMAP

iMAP, short for Illinois Mapping, takes schema matching further by finding complex matches [23]. Additional input consists of domain knowledge, including integrity constraints from the domain and the schemas, overlapping data, and external expert information. The system returns the top-k results, where most of the correct matches are [23].

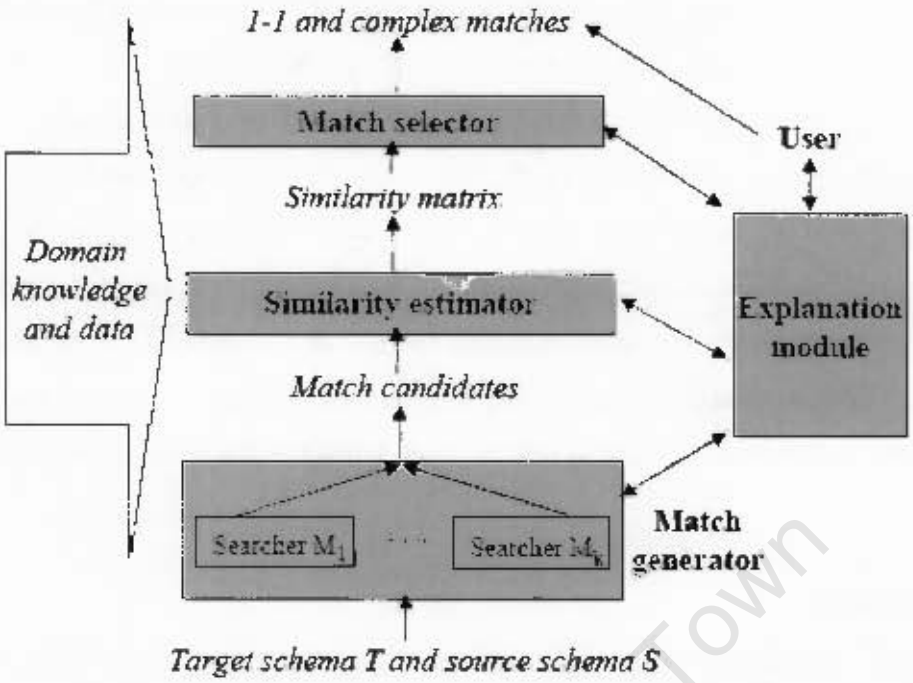


Figure 9: iMAP architecture (taken from [18])

Join paths are found before the match process is started. The architecture is made up of three parts, as shown in Figure 9. The match generator is responsible for managing the searchers, i.e. matchers. The similarity estimator combines the match scores to take into account other matches that were found, and domain and schema information. Two evaluator modules are used to do this: a Naïve Bayes evaluator and a name-based evaluator [18]. The name-based matcher lowers the match rank if the names are not similar enough. Finally, the match selector uses domain knowledge, like integrity constraints, to choose the final matches.

A new feature that was introduced by iMAP is the explanation facility [18], which allows the user to know how a match was created, how it passed the evaluation and selection process, which components were involved in determining the match, and what important assumptions were made while generating the match.

iMAP has seven searchers, as presented in Table 2. Each uses a set of heuristics to decide if it should examine the current attribute, e.g. the ratio between the number of

numeric and non-numeric characters and the average number of words per stored value can be used.

Table 2: Current implemented searchers in iMAP (taken from [18])

Searcher	Space of candidates	Examples	Evaluation Technique
Text	Text attributes at the source schema	name = concat(first-name,last-name)	Naïve Bayes and beam search
Numeric	User supplied matches or past complex matches	list-price = price * (1+tax-rate)	Binning and KL divergence
Category	Attributes with less than t distinct values	product-categories = product-types	KL divergence
Schema mismatch	Source attribute containing target schema info	fireplace = 1 if house-desc has "fireplace"	KL divergence
Unit conversion	Physical quantity attributes	weight-kg = 2.2* net-weight-pounds	Properties of the distributions
Date	Columns recognized as ontology nodes	birth-date = b-day / b-month / b-year	Mapping into an ontology
Overlap numeric	Specified by a context free grammar	interest-earned = balance * interest-rate	Equation discovery (LAGRAMGE)
Overlap version of the text, category, schema mismatch and unit conversion searchers (see Section 4 "Exploiting Domain Knowledge")			

In addition to heuristics, external data found in the domain may be exploited [18]. iMAP may mine real estate listings from the Internet to learn that the number of real estate agents in a specific area is bounded by fifty. iMAP can realize that if, for example, the concatenation of *firstName* and *lastName* yields hundreds of distinct names, it is unlikely to match *agentName*.

An iterative algorithm is used in which only the top-k highest scoring match candidates, where k is a pre-defined constant, are used in the next iteration. The algorithm terminates when the highest scoring match candidate at the end of the current iteration is beyond a given threshold.

Experimentation

Testing was done on four domains. Schemas with both disjoint and with overlapping data were used. They used a top-3 matching accuracy measure, which counted the fraction of target attributes whose top three candidates included the correct match.

With the schemas that included overlapping data, iMAP achieved a high matching accuracy of 68-92% [18]. The default iMAP setup achieved an accuracy of 58-74%, whilst exploiting domain constraints or overlap data further improved the accuracy by 12-23%. By exploiting both domain constraints and overlap data, the accuracy improved by as much as 11%. With the schemas that excluded overlapping data, iMAP achieved a matching accuracy of 62-79%. The default iMAP setup achieved an accuracy of 55-76%, whilst exploiting domain constraints improved accuracy by 9%.

When discovering only complex matches, the top-1 accuracy was 50-86% for the four overlap domains. The default iMAP configuration achieved an accuracy of 33-55% on all domains. By exploiting domain constraints, the accuracy was improved by up to 17% and by exploiting overlapping data, the accuracy improved by up to 46%. Exploiting both domain constraints and overlapping data improved the accuracy by 10-64%. Finally, the reuse of previous matches improved the accuracy by 28%.

Overlapping data improves the accuracy of the matches. Since it commonly occurs in real-world databases, it should be exploited more. The searchers were not able to detect small variations between attributes when building formulas. Either extra noise was added to the formulas or attributes that contributed little were left out, leaving the formula slightly incorrect. This did show that the searchers were very good at finding partial complex matches that the user could then complete. The systems took between five and twenty minutes to run.

3.4.8 Summary

All of the aforementioned systems require some form of pre-match effort from the user. LSD provides extensive instance-level matching. Cupid was the first to introduce datatype matching, which is now used by the majority of systems. Similarity flooding is the only system that does not require any additional input, besides the schemas. COMA is a composite system that contains numerous matchers. It is the only system to have done extensive testing (see Table 3). SemMa is the only system to use a web service (WordNet) as a dictionary. MKB focuses heavily on exploiting past matches. Its use of a Bayesian Network allows it to learn and therefore improve over time. Finally, iMAP is the first system to find complex matches and to provide explanations for the matches. All systems mentioned that the scalability of their approach is an issue and that they had not considered factors such as execution speed.

Table 3: Overview of schema matching evaluation scores

	LSD	Cupid	Similarity Flooding	COMA	SemMa	MKB	iMAP
Average Recall	-	-	-	0.89	-	-	-
Average Precision	0.82	-	-	0.92	-	-	-
Average F-Measure	-	-	-	0.9	-	-	-
Average Overall	-	-	0.6	0.82	-	-	0.8

Table 4 provides an overview of the systems and illustrates several trends. For instance, the use of the similar neighbour assumption is common. The name matcher is very popular as is the datatype matcher. Of the seven systems, five combine match candidates using the weighted sum, whilst Similarity Flooding uses a threshold and COMA averages the scores.

Table 4: Summary of different schema matching systems

	LSD	Cupid	Similarity Flooding	COMA	SemMa	MKB	iMAP
Date	2001	2001	2002	2002	2003	2003	2004
Input includes domain information	Yes	Yes	No	Yes	No	No	Yes
Output cardinality	1:1	1:n	1:1	1:1	1:1	Unspecified	m:n
Number of matchers	5	3	3	13	3	5	7
Structure matcher	Yes	Yes	Yes	Yes	Yes	Yes	No
Use similar neighbour assumption	No	Yes	Yes	Yes	No	Yes	No
Name matcher	Yes (custom dictionary)	Yes (custom dictionary)	No	Yes (custom dictionary)	Yes (WordNet)	Yes (No dictionary)	No
Edit distance matcher	No	No	Yes	Yes	No	No	No
Datatype matcher	No	Yes	Yes	Yes	Yes	Yes	No
Instance-level matchers	Bayesian Networks, Whirl	None	None	None	None	Bayesian Networks	Bayesian Networks, KL divergence
Reuse	Yes	No	No	Yes	No	Yes	Yes

3.5 P2P Database-Sharing Systems

This section covers schema matching in P2P database-sharing systems.

3.5.1 Edutella

Edutella uses a mediated schema to represent the schemas on the network. Peers are required to define how their schema maps to this mediated schema manually [42]. This must be written in RDF, and ontologies can be used, but they only provide limited matching help. In addition, very few schemas are present in a format that can be directly used by these tools, hence the need for manual intervention.

3.5.2 Piazza

Piazza is a P2P database-sharing system that uses pairwise mappings to map pairs of peers' schemas together. Three combinations of mappings can occur [31]:

- Pairs of peers with XML Schemas
- Pairs of peers with RDF schemas or OWL ontologies
- One peer having an XML schema and another with an RDF schema or OWL ontology

Mappings between schemas are defined manually by domain experts [29]. The most popular mechanism for describing mappings in data integration is to use views [31]. There are two ways of using views. With the first technique, known as global-as-view, the mediated schema is a set of views over the data sources. The second technique, called local-as-view, describes the data sources as views over the mediated schema. This requires more sophisticated query reformulation algorithms, but offers more flexibility. Piazza combines and generalizes the two data integration formalisms [29].

3.5.3 Hyperion

This system uses a combination of mapping tables, expressions, and functions instead of views to achieve data integration. All mappings are currently created manually by domain experts. Hyperion uses event-condition-action (ECA) rules to enforce mapping constraints, including mapping expressions. Views require peers to be willing to share their schemas and cooperate in establishing and managing queries. Such close cooperation in P2P environments can be undesirable, perhaps for privacy reasons, or

unfeasible, perhaps due to resource limitations or the dynamic nature of the data structures [36].

Mapping expressions are based on the work proposed in [9], particularly the Local Relational Model (LRM). This model enables general queries to be translated into local queries, using the concept of local coordination formulas to translate between schemas. The model has two useful properties. It supports inference of new mappings from existing mappings. Mappings can also be used as constraints to identify valid and invalid mappings between attributes that reside on different peers [36].

3.5.4 BestPeer

BestPeer uses a simplistic method to achieve schema matching. It relies on each schema attribute being described by keywords. For example, a peer may describe the schema attribute *ProtSeq* using the keywords protein, sequence, and number [44]. These keywords are then used, along with attribute names, to match attributes using simple string comparisons. A problem with this is that peers may describe the same attribute using different keywords; yet, they claim that the approach is quite accurate [41].

3.6 Summary

A vast range of schema matching techniques and tools has been described. In the context of P2P database-sharing systems, however, schema matching is usually done manually [42, 31, 36], or else kept simple and requires special pre-match effort [41].

Chapter 4: Design

This chapter presents the overall design of Mapster; a more detailed discussion of its main components follows in chapter 5.

4.1 Overview

Effective schema matching techniques have been developed, but typically focused on static and controlled environments. Several issues may cause existing schema matching approaches to perform poorly in P2P networks, 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 issue important. In addition, there is no central authority, which makes it difficult to keep an up-to-date picture of all the current peers' schemas. P2P systems that have provided database-sharing services [42, 31, 36, 41] have focused more on what can be done once the mappings are in place and not on how to determine the mappings. Edutella is one of the more sophisticated systems, which provides many services, such as querying and caching of the mediated schema. It uses HyperCuP, a very efficient super-peer topology, to enable efficient querying. However, only limited mediation is possible using their approach. BestPeer offers schema matching, but this is only basic string matching that is done on attribute keywords, which must be supplied by the users.

My solution has focused on these aspects and has resulted in a P2P system, called Mapster, which focuses on semi-automatic schema matching. Unlike other P2P database-sharing systems, Mapster offers a variety of schema matching functions that do not require domain knowledge to achieve schema mediation. The approach is not fully automatic as user validation is included. Determining the mappings semi-automatically allows the mappings to be more accurate, whilst requiring less human effort, for example, by avoiding pre-match preparation of the databases and input from domain experts. Previous chapters highlighted several approaches to the various problems that this work is trying to address. No single system addresses all problems sufficiently.

Mapster exploits the topology of the P2P system to make schema matching viable. I am not aware of any other P2P database-sharing system that has focused on the topology for this purpose. Mapster uses a super-peer network and clustering, according to peers' particular areas of interest or domains. A super-peer network was used as it provides a structured environment where communication is efficient. Super-peer topologies have become popular in newer systems, as they address the problem of efficiency within the network. This is important, as cooperation between peers is necessary for them to share lots of (meta)data effectively. Clustering peers according to their domain helps to break the network up into sections that are more manageable. Many systems use clustering to group peers with similar interests together to improve query results and processing time. It also helps increase the chance that peers' schemas contain overlapping data, which is beneficial to the schema matching process [62]. These domains are individually more stable than the entire network and, so, provide an environment that is better suited to existing schema matching techniques.

Clustering creates domains that comprise many databases storing related information. However, these databases will certainly be structurally different. There are two common methods of handling these differences, i.e. enabling the transformation of information from one database to another with a different schema. This transformation ability is needed so that the various databases of the domain, and the network, can be queried by any peer, irrespective of their own database. The first method is to use pairwise mappings and the other option is to create a mediated schema. Both approaches have been used before, for example, Piazza uses pairwise mappings whilst Edutella uses an approach synonymous with a mediated schema.

A pairwise mapping is a mapping from a combination of attributes in one database to a combination of attributes within another database [31]. 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. Nevertheless, these mappings can be used to determine new transitive mappings, making pairwise mappings quite flexible. 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 and query processing problematic when pairwise mappings are used.

A mediated schema is a single schema that is the combination of several schemas [31]. A mediated schema can be used to represent the schemas of all peers that have recently[8] joined a domain. 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 peers' schemas, thus the mediated schema approach is simpler and more scalable. Query processing is easier as the query only needs to be translated once for each peer, being from the mediated schema to the relevant peer's schema. A disadvantage to using mediated schemas is that it can be difficult to maintain an up-to-date version in a P2P network if it is not implemented properly.

Based on its advantages, the mediated schema approach appears preferable to the pairwise mappings method. A single mediated schema is maintained for each domain. The creation and maintenance of the mediated schema is done by the super-peers. Super-peers, which are discussed in section 2.3.3, have more resources and are more stable than normal peers are. This makes them good candidates for managing the mediated schemas. Each domain contains a super-peer, which constructs the mediated schema for the domain from the schemas of the peers that have been connected to the domain. If a peer wishes to join the domain, it first communicates with this super-peer, which will add the new peer's schema to the mediated schema, using schema matching techniques. The super-peer also performs other functions, which are mentioned later on in this chapter.

Super-peers from different domains can communicate with each other to support global queries, i.e. queries that are not specific to any single domain. This is useful if a user wants to combine information from various domains. 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.

[8] Mappings are removed from peers that have not been online for a while, e.g. a month

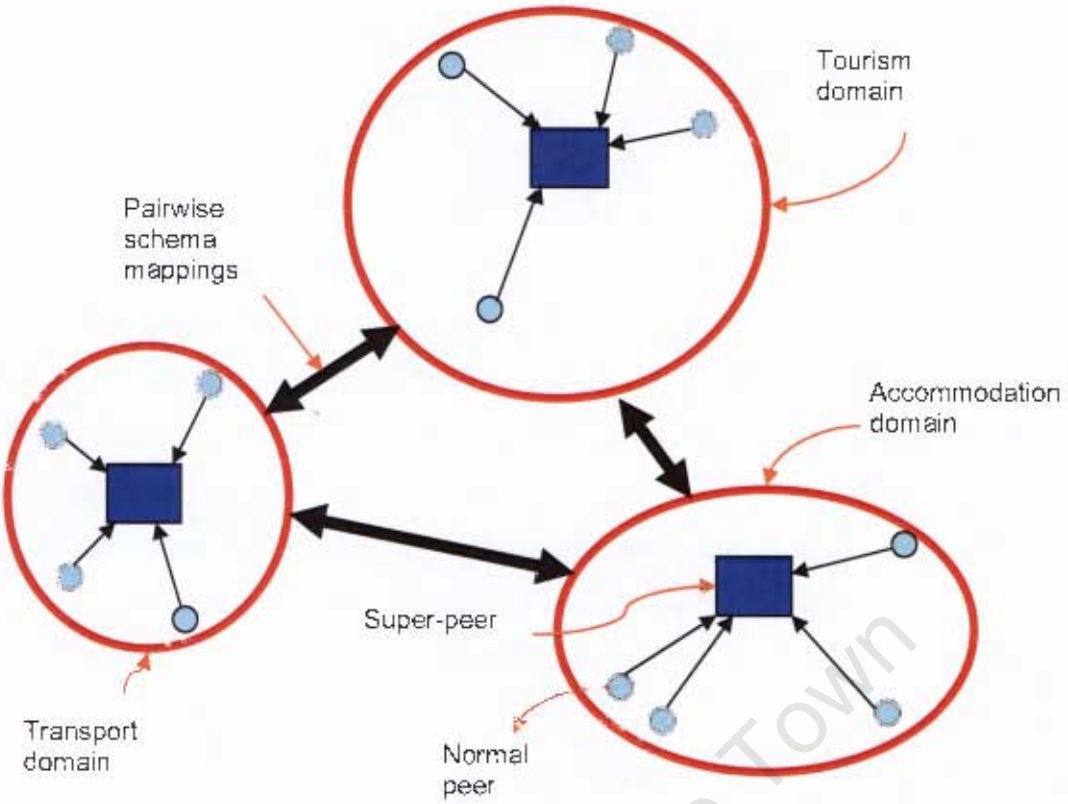


Figure 10: Basic Mapster Network

Figure 10 illustrates a basic Mapster network. The large circles represent domains. Each domain contains a single super-peer, represented by a large square, and several normal peers, represented by small circles. The thick arrows indicate the connections between domains. Consequently, both a mediated schema, which is used within a domain, and pairwise mappings, which are used between domains, is found in Mapster. This allows Mapster to benefit from the advantages offered by both approaches. The design of the Mapster components is discussed in the remainder of this chapter.

4.2 Components

The software that resides on each Mapster peer comprises several components, which are shown in Figure 11. The JXTA kernel, shortened to JK, forms the foundation of the software. It is responsible for all P2P connectivity, behaviour and communication. Above the JXTA kernel is the topology manager, shown as TM. The super-peer topology is managed by this component. Two components sit on top of this one, being the mediated schema constructor (MSC) and the query processor (QP). The mediated schema constructor creates and maintains the mediated schema for a domain. It is only used by super-peers. The mediated schema constructor has a subcomponent called the

schema matcher, shown as SM. The schema matcher is responsible for determining the mappings between database schemas. It employs several schema matching techniques, coupled with user interaction, to achieve this. The query processor, which is also only used by super-peers, handles tasks like translating a query between a peer's schema and the mediated schema and generating peer-specific queries from the original query. Finally, interaction between the user and the Mapster is handled by the user interface. It allows the user to join a domain using the schema matcher, connect to a domain, view the mediated schema, write queries, and view results. The rest of this chapter outlines each of the components in turn.

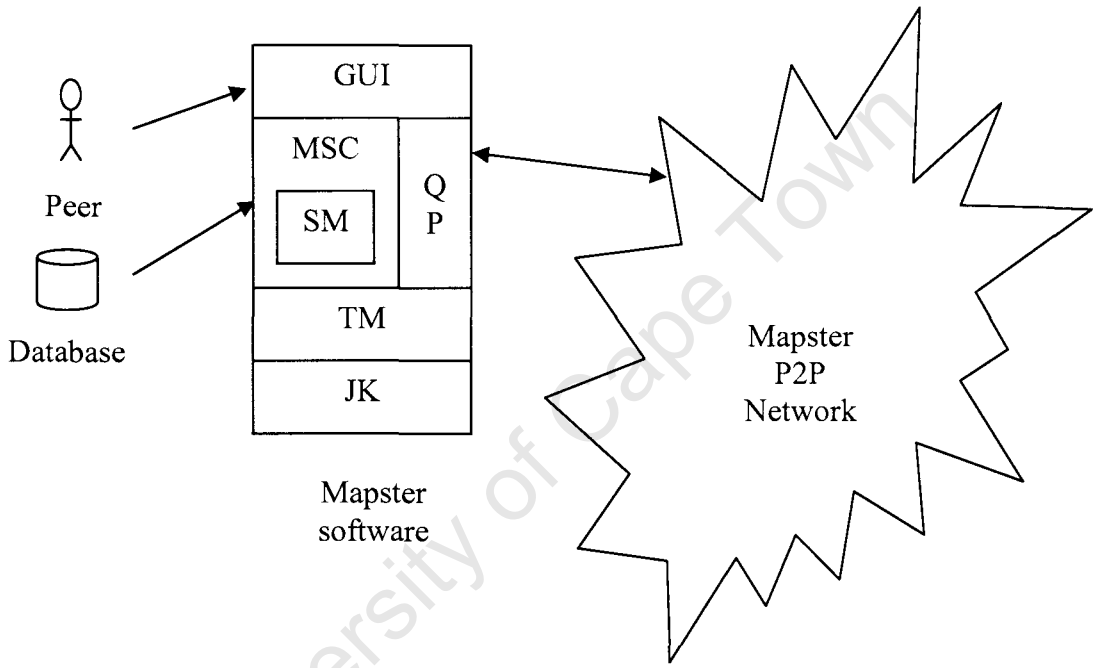


Figure 11: Overview of components

4.2.1 JXTA Kernel

The JXTA framework is a set of open and generalised P2P protocols and primitives that can be used to build a P2P network [11]. It allows any connected device on the network, from cell (mobile) phone to server, to communicate and collaborate as peers [26]. Its goal is to develop the basic building blocks and services needed to enable innovative P2P applications. JXTA technology seeks to overcome potential shortcomings in many of the existing P2P systems, including interoperability, platform independence and ubiquity. Currently, there is a stable and widely supported Java implementation of JXTA, which addresses the issues of platform independence and ubiquity.

JXTA was used to create the underlying P2P system, as it provided all the basic P2P functionality and allowed the more important aspects, like schema matching, to be focused on. The finer details of the JXTA kernel, including P2P communication and coordination, are explained in section 5.1.

4.2.2 Topology Manager

The super-peer topology localises network changes and reduces the effect these changes have on the surrounding peers. It also pushes unstable peers to the edge of the network, where the effects of their dynamic behaviour are minimised. Unstable peers connect at random times for varying amounts of time. This dynamic behaviour could cause considerable disruption if the peers were allowed to become super-peers, as the domain would have to restructure itself every time one came online or went offline.

Section 4.1 explained that a single super-peer is used to manage a domain containing normal peers (NPs). A normal peer is any peer that is not a super-peer. However, having a single super-peer 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 was introduced by having several super-peers present within a single domain. Multiple super-peers, each with its own distinct set of NPs, also allow for load balancing and for fault tolerance. These super-peers all store the same mediated schema so that when one super-peer goes down, the mediated schema is still available. Queries can be distributed amongst the super-peers to achieve load balancing. Since the most reliable NPs are made super-peers, super-peers are unlikely to go offline excessively in any case.

Section 2.3.3 gave a list of requirements that need to hold for a super-peer topology to work effectively. Two of these requirements are met by implementing multiple super-peers in a domain. Firstly, redundancy is provided by replicating the mediated schema amongst the super-peers. Redundancy is used to avoid issues such as having a single point of failure, like a central sever, and improves scalability. Secondly, network traffic, and peer workload, during query processing is distributed amongst the super-peers. This is achieved as each super-peer has a distinct set of NPs. These NPs only need to query their parent super-peer as it always has an up-to-date copy of the mediated schema. A third requirement is met using a property defined in section 5.2, called NP acceptance.

This defines how many NPs can connect to a super-peer and enforces the requirement that the number of neighbours for any peer must be limited.

Control in a domain is maintained by having a certain super-peer act as the root of the domain. This super-peer performs the bulk of the decision-making. When peers first connect to a domain, this is the first peer with which they communicate. It is responsible for maintaining the mediated schema and distributing it amongst the remaining super-peers in that domain. This is the super-peer that super-peers from other domains contact when querying that domain's mediated schema and is called the virtual super-peer (VSP). All super-peers present within the domain are called actual super-peers (ASPs). This means that the VSP is also an ASP. All ASPs are also NPs. Each ASP is-a NP and the VSP is-a ASP.

Some examples of how the topology works are presented next. Figure 12 demonstrates the super-peer topology without super-peer replication, i.e. without ASPs, whilst Figure 13 illustrates the topology with super-peer replication. Figure 12.a shows a domain just after a peer created it. Since the peer started the domain, it becomes the super-peer for it. This association is shown by the thick arrow. Notice that it is also an NP. The mediated schema is created using the peer's database. Figure 12.b shows what the topology looks like after more peers have joined the domain. If an NP goes offline, then the super-peer simply updates its list of online peers and flags the NP's database as offline in the mediated schema. However, if the super-peer goes offline, as shown in Figure 12.c by the dotted boxes, then a new super-peer must be chosen and the mediated schema needs to be reconstructed. The resulting topology is shown in Figure 12.d.

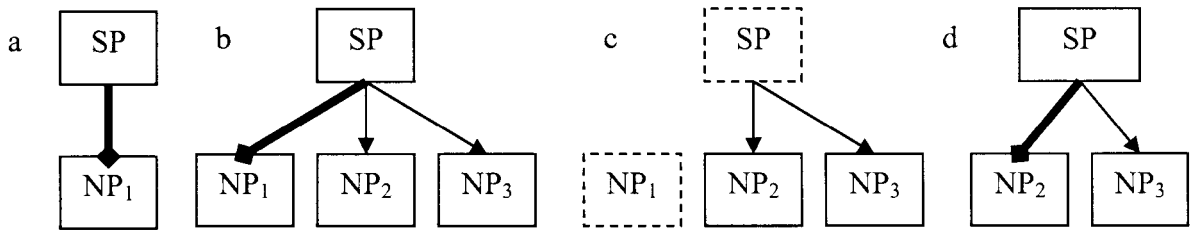


Figure 12: Super-peer topology without super-peer replication

The examples in Figure 13 illustrate how the addition of ASPs adds a level of indirection between the VSP and the NPs. Two new scenarios occur with replication,

namely when the VSP is changed and when an ASP is changed. If the VSP goes offline, shown in Figure 13.c, then a new VSP is chosen from the existing ASPs. The new VSP uses its copy of the mediated schema as the new domain mediated schema. Once the old VSP's database has been flagged as offline in the mediated schema, the new VSP distributes the mediated schema to all other ASPs. This is far better than having to reconstruct the mediated schema, since using schema matching to construct a new mediated schema from all the peers' databases is time consuming. If an ASP goes offline, then its children NPs ask the VSP to add them to another ASP. This is shown in Figure 13.d, where the second NP from the first ASP connects to the only remaining ASP. This three-layer approach is more robust and better suited to the P2P environment. It reduces restructuring within the domain thus creating a more stable workplace for the schema matching component.

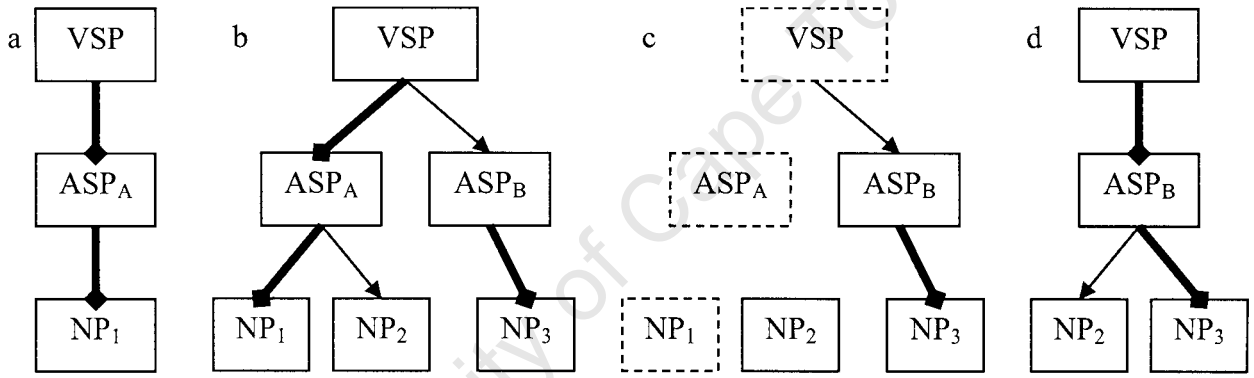


Figure 13: Super-peer topology with super-peer replication

4.2.3 Mediated Schema Constructor

The mediated schema construction algorithm is incremental, which allows a database to be added at any point with minimal interference to the working of the mediated schema. The algorithm was adapted from the work done in WISE-Integrator [33]. The mediated schema begins empty. A database is added by taking each schema attribute and placing it in a cluster within the mediated schema. Clusters are sets of schema attributes that are semantically equivalent. A cluster in the mediated schema is analogous to an attribute in a schema, such as *Price*. Once the first database has been added to the mediated schema, other databases can be added following a similar procedure. However, the relevant cluster in the mediated schema must be found for each new schema attribute. This is where the schema matching techniques are used. The construction and maintenance of the mediated schema is explained in section 5.3.

An example of how the mediated schema is constructed is now given. The following tables illustrate three simple, yet slightly different, schemas that have been defined for this example. The mediated schema is initially created from schema A. Since the mediated schema is empty, each attribute from schema A is put into its own cluster. The entries in the clusters are in the form of Schema.Attribute. The relation name has been left out of the tables for brevity. In step 2, schema B is added to the mediated schema. Schema matching is used to decide to which cluster each new attribute belongs. After schema C has been added, shown in step 3, one can see how the mediated schema represents the combination of the three schemas. Notice how a new cluster was created for the attribute *Age* and how each cluster only contains semantically equivalent attributes. For instance, *Age* was not put into cluster one, even though its stored values are similar to those of cluster one.

Table 5: Schema A

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

Step 1: Mediated schema is created using schema A

Cluster 1 = {A.ID}

Cluster 2 = {A.FName}

Cluster 3 = {A.LName}

Table 6: 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 7: 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}

4.2.4 Schema Matcher

Schema matching is used in the construction of the mediated schema. In order to add a new database schema to the mediated schema, the mappings from the schema's attributes to the clusters within the mediated schema need to be defined. These mappings are found semi-automatically using schema matching techniques. These techniques allow the mediated schema to be constructed far more quickly and effortlessly than manually defining it. Once the mappings have been adjusted and confirmed by the user, they are added to the mediated schema for later use by the query processor. Mediated schema updates are only done when a new database joins a domain or when an existing database undergoes schema changes, i.e. the schema mappings are not lost when a peer goes offline, as this saves time by skipping the schema matching phase when the peer reconnects.

The schema matcher is a composite matcher that is made up of six individual matchers. There are two schema-level matchers, being the name path matcher and the Similar Neighbour matcher. Three element-level matchers have been implemented, which are

the WordNet matcher, edit distance matcher and a datatype matcher. A keyword matcher has been used as the instance-level matcher. These matchers are covered in detail in section 5.4. These matchers were chosen based on the research provided in chapter 3. Besides some of the more complex matchers, these were the matchers used in the majority of the systems. The keyword matcher was used instead of a machine learning approach to save implementation time and reduce the complexity of the instance-level matcher.

The way in which these matchers are selected, tuned and combined is specified in a configuration file, which is created, with defaults that can be modified for a specific domain, using a utility covered in section 5.4.1. In addition to allowing the matchers to be fine-tuned, this utility allows the match candidates to be combined in one of four ways: weighted sum, average, maximum, and minimum. This covers all possible combination methods, as described in chapter 3. The schema matching process is presented in Figure 14.

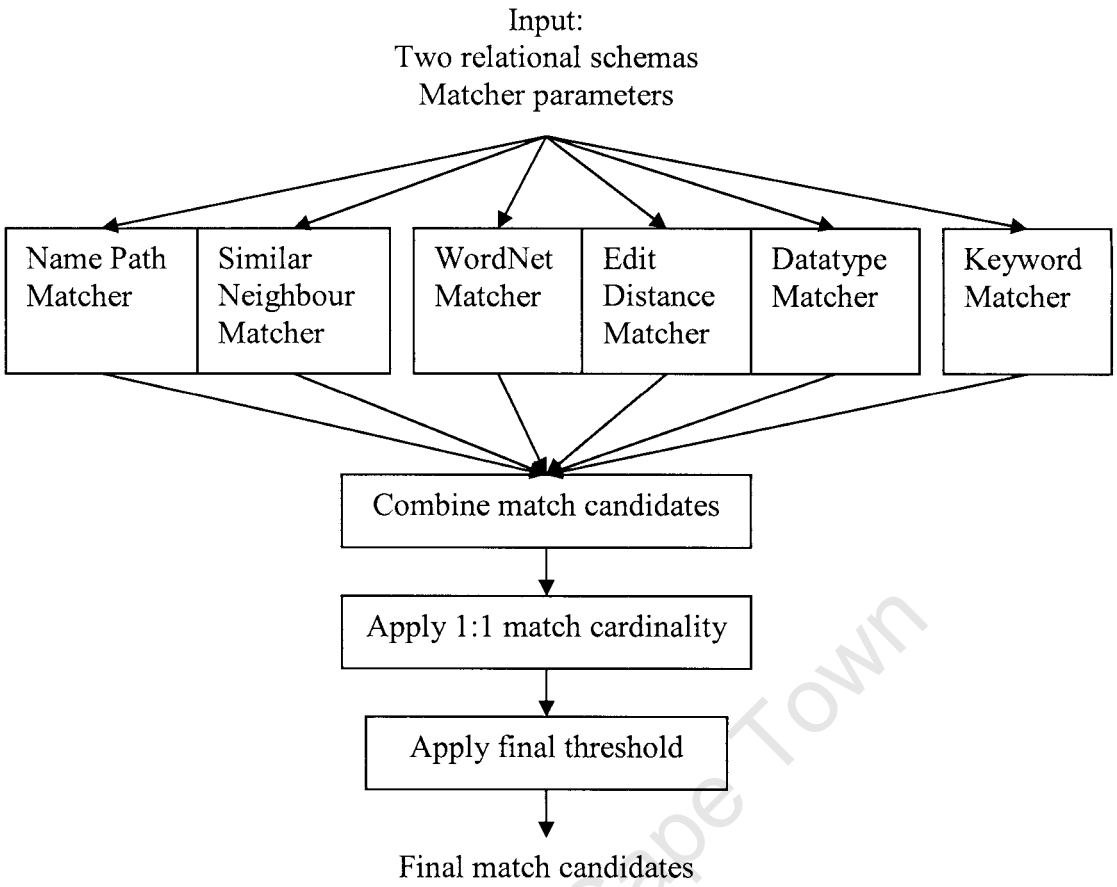
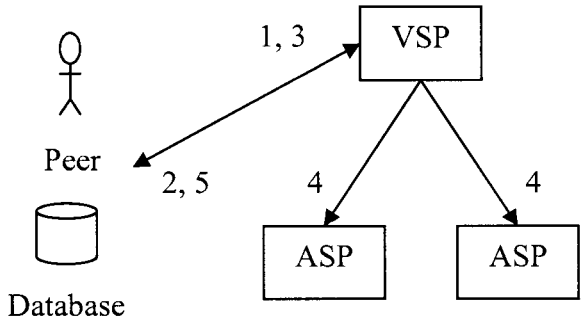


Figure 14: Overview of schema matching process

The match process is run at the VSP. The input schemas are the mediated schema and the new peer's schema. Each matcher runs separately to produce its own set of match candidates. These individual match candidates are then combined and sent back to the peer for validation. The peer can adjust them or add new ones. The final mappings are then sent back to the VSP, which adds them to the mediated schema. The new mediated schema is then distributed amongst the ASPs. The process is summarised in Figure 15.



1. Send database schema to VSP
2. Receive match candidates from VSP
3. Confirm matches and send final mappings to VSP
4. Add mappings to MS and distribute it amongst the ASPs
5. Send acknowledgement back to peer, to confirm that mappings have been added to the MS

Figure 15: Schema matcher involvement in Mapster

4.2.5 Query Processor

Querying processing is the last step in achieving a P2P database sharing system. Once a peer has joined a domain and had its database added to the mediated schema, 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. This is impractical in a large environment, such as a P2P network. Users can browse the mediated schema to achieve a similar effect, although the stored values of the schemas cannot be viewed from the mediated schema. Querying is far more powerful, as it allows the user to specify which schema attributes he wants to view, what range of values they should fall into and how they should relate to other attributes, either in the same database or in other databases. Queries, by default, are specified in terms of the peer's own database schema. If this is not adequate then the query can be specified in terms of the mediated schema instead. Queries written in terms of the local schema are preferred as the user should understand his database better than the mediated schema and, consequently, be able to specify a query more accurately. However, using the mediated schema instead can provide access to more attributes of the domain, which could be absent from the local schema.

Once a query has been specified, it is sent to the peer's parent ASP. Here, the query is first rewritten in terms of the mediated schema if necessary. This requires the ASP to replace attribute names in the query by corresponding cluster names from the mediated schema. The ASP then uses the mediated schema mappings to generate database-

specific queries, called Peer Queries, for each peer that can meaningfully contribute to the result. These Peer Queries are then sent to the applicable peers, which send results back directly to the peer that posed the query.

4.3 Summary

Mapster uses the concept of super-peers to structure a P2P network appropriately for database sharing in the presence of heterogeneous database schemas. Clustering by domain is used to group similar schemas together to ensure overlapping data is present and consequently improve the schema matching. Virtual Super-peers (VSPs) are responsible for semi-automatically maintaining a mediated schema for their domain. A chained architecture links these *domain servers* together. To improve fault tolerance and performance, each VSP has its own Actual Super Peers (ASPs) where the domain schema is replicated and query translation is done. Incremental logins rather than batch logins was considered best for database sharing, and the VSP/ASP architecture combines the benefits of a chained architecture with those of full replication.

Chapter 5: Implementation

The implementation of Mapster was broken up into several parts. Each major section was first prototyped, allowing the concepts to be tested and refined before being integrated to form Mapster. The six main components of Mapster introduced in chapter 4 are described next, along with any prototypes that were developed.

5.1 JXTA Kernel

The JXTA kernel is responsible for all P2P communication and functionality. Basic P2P operations were performed by the JXTA library, which were then built upon to offer functions that were more advanced. Although the functions are not complex, their interaction is complex and the implementation proved to require lots of work and effort; far more than the schema matching component. In addition to the functions mentioned below, JXTA also handles the login process, so that each peer can be assigned a unique JXTA ID. This ID is used for communication purposes.

5.1.1 Adverts

JXTA uses adverts to allow peers to broadcast information over the network. An advert is an XML document that names, describes and publishes the existence of a resource [25]. Two custom advert types were made for Mapster, being the *Domain Advert*, which is mentioned in 5.6, and the *New VSP Advert*, which is explained in section 5.2. Once a domain is running, the VSP will periodically update and publish the *Domain Advert* to notify other peers of its existence and status. This status includes information like the number of peers in the domain, subject of the domain, uptime, and the ID of the VSP. The ID is used to communicate with the VSP. As this status can change often, a domain advert is currently published every 30 seconds. A single advert with a time-to-live value would not suffice. The way in which it is published is handled by the JXTA Kernel.

5.1.2 Pipes

The JXTA library uses pipes to allow peers to communicate with each other directly. Pipes are communication channels used for sending and receiving messages [26]. They are unidirectional, so there are input pipes and output pipes. Bidirectional pipes were not used as they become problematic when a peer has several pipes open. Problems with synchronisation and port blocking occurred. Unidirectional pipes were much easier to

use as they could be closed as soon as the all the data had gone through the pipe, which freed up the port bindings. Pipes are extensively used for all direct[9] communication between peers. Each peer has two managers called the Output Pipe Manager and the Input Pipe Manager. The Output Pipe Manager is used to open pipes with other peers and then to send messages down the pipes. The manager is also used to test if a peer is online. This is done by trying to open a connection to it. If this fails after a specified time, then the peer is labelled as offline. The Input Manager is started as soon as the peer is started and uses a server pipe to continually listen for incoming pipe connections. This server pipe runs in its own thread so that it does not interfere with the normal operation of the peer. Once a connection is made with another peer, the server pipe will create a new pipe between those two peers and continue listening for new connections.

5.1.3 Pinging

In order to have up-to-date peer information, pinging functionality was added to the JXTA kernel. Research into the JXTA library showed that their solution of having up-to-date peer information was only a proposal and not yet implemented [11]. Consequently, a pinging component was added to Mapster to address this.

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. The current timeout settings are as follows: an NP must ping its parent ASP every 15 seconds and the ASPs must ping the VSP every 20 seconds. If the 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 remove from the list of children on the ASP. The same applies to the VSP pinging ASPs. If a peer cannot ping its parent then it will try to ping the VSP, assuming that the parent of the peer is not 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. This is

[9] Adverts are used to broadcast information to the entire network, whereas pipes are used to communicate information directly between two peers

discussed in section 5.2. These checks allow the peers to handle the dynamic nature of the network, where peers do come and go at random.

Each pinging component is run in its own thread. Again, this is to prevent it from interfering with the normal operation of the peer. It also allows the peer to perform several tasks at once. If the peer is an ASP, then it will have two pinging components running; one to ping the VSP and another to ping the NPs connected to it.

5.1.4 JXTA Database-Sharing Prototype

Before topology management was implemented, a prototype system was created to test JXTA's ability to share databases. A custom JXTA advert type was created called a database advert and contained information such as the peer's ID and an outline of its schema. Whenever a new peer connected to the JXTA network, a database advert for the peer's database was published, and a copy of the advert saved in the peer's local cache. This allowed another peer to retrieve the advert from that peer's local cache[10] at any time, assuming that it was still valid, as adverts contain a timestamp that expires. Once a peer has obtained another peer's database advert, it can use the advert to view and query the peer's database. Figure 16 illustrates how the prototype worked.

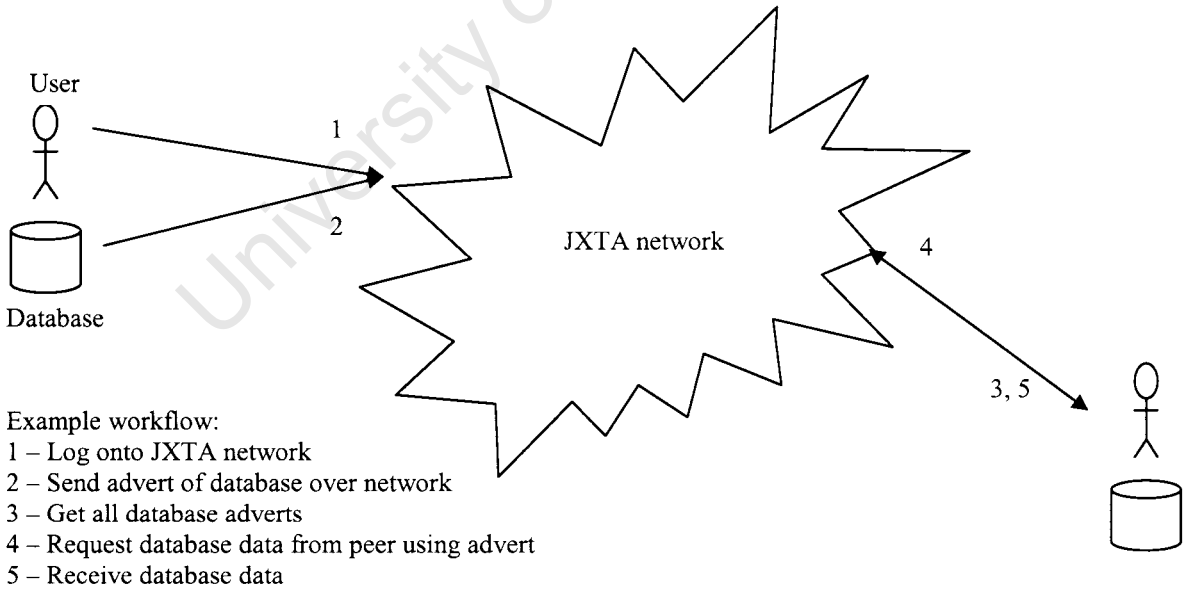


Figure 16: JXTA database sharing outline

[10] Note that security and privacy issues are outside the scope of this research

5.2 Topology Manager

The super-peers' topology managers all work collaboratively to manage what happens within a domain, although the VSP's topology manager is in charge. The network as a whole is not managed, as there is no central authority. This allows domains to be created and to disappear at will. The VSP's topology manager decides where new peers are connected within the domain so that the load on the super-peers is balanced. It handles replication by deciding which peers should become ASPs.

Each peer is able to define its own ASP behaviour such that when it becomes an ASP and too many NPs try to connect to it, a new ASP would be created from the most willing NP. NP willingness to become an ASP is calculated using the peer's information, such as average CPU load, network behaviour, bandwidth capabilities, etc. This value is called the SP willingness. The user can also set a default value for the node's SP willingness. This value indicates if the user would like to have the peer become an ASP or definitely not become an ASP. However, if the peer is the only one present within a domain, then it will become the VSP irrespective of its current SP willingness. These values are stored on the ASP with the list of NPs connected to that ASP.

New peers are then connected to the most accepting ASP. The degree to which ASPs are accepting of new peers is based on information like the current number of connected NPs and the maximum number of NPs that the ASP is willing to accept. This acceptance value can be adjusted by the user and is called NP acceptance. The VSP stores these NP acceptance values with the list of currently connected ASPs.

The manager aims to build a balanced tree, where the VSP is the root and the NPs are the leaves. However, each ASP can adjust its NP acceptance value, and SP willingness values change over time for every peer. NP acceptance will only change when a user changes it, which is typically very rare. SP willingness, on the other hand, may change often, as it is a measure of the workload of the peer. These two factors often mean that the topology is not structurally balanced, but balanced in terms of load on the super-peers. This is useful for scalability.

The ability to calculate how willing a peer is to become an ASP is useful in stabilising a domain. Since only the most willing peers become ASPs, ASPs within a domain should be quite stable. Every time a new peer joins the domain, its willingness to be an ASP is calculated. If it is substantially more willing than an existing ASP then it replaces the least willing ASP. This ensures that the domain remains stable in the long term, by sacrificing some stability in the short term.

The VSP is not changed unless it goes offline. If the VSP does go offline then the ASPs will stop whatever they are doing. They will each publish an advert, called the *New VSP Advert*, which stores the fact that they are willing to become the new VSP. It also stores their SP willingness. Each ASP will then listen for similar adverts from other ASPs. Every time such an advert is received from another ASP, the SP willingness is compared to the current ASP's SP willingness. The higher of the two is then stored along with that ASP's ID. This SP willingness is then compared to other ones that are received in adverts. After a given amount of time, currently set at 15 seconds, this process is stopped and the ASPs check the stored ID of the most willing ASP. If the ID matches an ASP's ID then it becomes the new VSP. The other ASPs will then start to listen for domain adverts. Once they receive domain adverts from the new VSP, they can reconnect themselves to the domain. This cooperation ensures that the most willing ASP becomes the new VSP.

The peer status is constantly updated and can be seen in the peer status panel, which is shown in Figure 17. It displays all the details of the peer and allows the user to adjust its SP willingness. The limit on the number of NPs that can connect to it can also be adjusted to change its NP acceptance value. Both of these can be done at runtime.

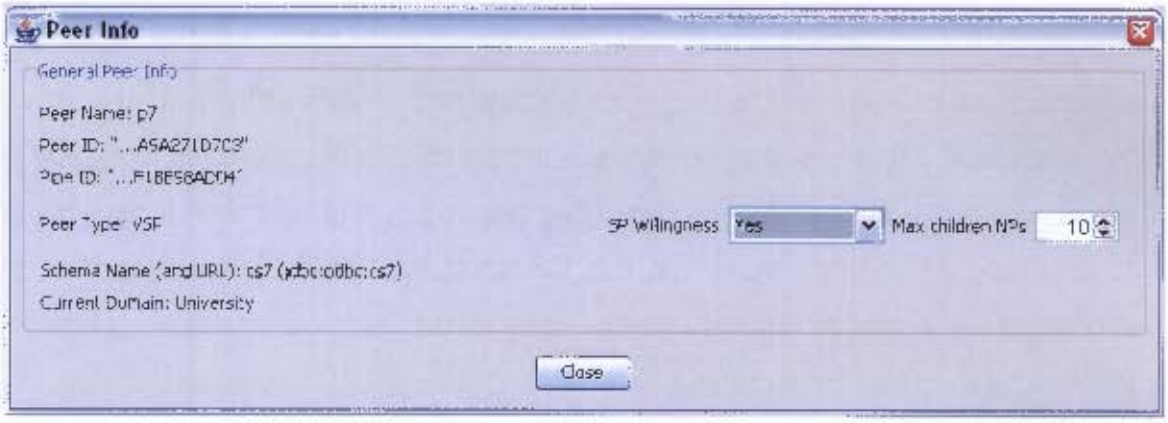


Figure 17: Peer Status Panel in Mapster

5.2.1 Super-Peer Topology Simulator

Before Mapster was changed to incorporate ASPs, a simulator was created to examine the topology creation and behaviour. This was primarily done to test how a super-peer would adjust to high loads. The simulator was responsible for the notions of SP willingness and NP acceptance in order to decide on how to adjust the ASPs load and where to put new NPs. The simulator also helped highlight the need to slow down the reordering process of the ASPs. Every time a new peer came online with a high SP willingness, the domain would restructure. It was decided that the new peer must be more willing to become an ASP than an existing ASP by 20% before a change is made. This helps to reduce ASP changes without affecting the performance.

5.3 Mediated Schema Constructor

The mediated schema is the combination of all schemas present in that domain. In order to combine schemas, schema matching is used to find which schema attributes are similar. These similar attributes are grouped together in clusters. Attributes that are not similar are then put into new clusters. Every time an attribute is added to a cluster in the mediated schema, a mapping is stored between the cluster and an object. This object contains information necessary to distinguish the schema attribute from all other attributes. It includes the peer ID, and the names of the schema, relation and attribute. These mappings are stored in a table, called the schema mapping table, and are used in query processing to link attributes in the mediated schema to attributes in the peers' schemas. This is necessary in order to redirect queries to the relevant peers.

The algorithm presented below outlines the mediated schema construction process. Once the mediated schema has been created for a new domain using the first database, which is explained in section 4.2.3, new databases are added by comparing each attribute in the database to all clusters within the mediated schema. To compare an attribute against a cluster, the attribute must be compared to all attributes within the cluster using schema matching. Schema matching between attributes is explained in section 5.4. The similarity scores between each attribute and the new attribute are then combined to represent a single similarity score between the cluster and the new attribute. Combination can be done by either calculating the average or by using the minimum or maximum score. However, using the maximum is too optimistic and too many clusters end up having a high similarity to the new attribute. Using the minimum value is too pessimistic and too few clusters, if any, are returned. Therefore, the average is used. The new attribute is then added to the cluster with the highest similarity. However, if the match similarity is zero for all clusters then the new attribute is put into a new cluster. The relevant mapping is also added to the schema mapping table. This process is performed for every attribute in the new database. Section 4.2.3 illustrates this process.


```
For every attribute a1 in the new database
   For every cluster in the mediated schema
  {
      For every attribute a2 in the current cluster
      {
          Compare a1 and a2 using schema matching
          Combine the match probability with the cluster score
      }
      c_max = cluster with the highest match probability
      If the match probability of c_max is not 0 then
          Add a1 to c_max
      Else
          Create a new cluster for a1
      Update the schema mapping table
  }
```

Maintenance is straightforward. Removal of schemas is achieved by marking all attributes from the relevant schema as offline. They are not removed to avoid having the peer go through the schema matching process again when it reconnects. The mappings are removed from the mediated schema if the peer does not reconnect after a specified time, currently set at a month. However, the mappings are still stored on the peer for any future use. Schema adjustments are done by removing the old mappings from the mediated schema and applying the schema addition process to the new schema attributes.

The mediated schema is constructed by placing each schema attribute into the relevant mediated schema cluster. This may change in the future, as clusters may not contain single schema attributes, but combinations of them, and future schema matching techniques may indicate that combinations or joins of attributes are more appropriate than single attributes. For example, if the mediated schema had a cluster that contained an attribute called *FullName* and a database was added that contained two schema attributes, *FName* and *Surname*, then it would be useful to join these two attributes and store the result in the same cluster as *FullName*.

5.4 Schema Matcher

The schema matcher is responsible for determining how a new peer's database maps to the mediated schema. It uses various schema matching techniques called matchers to determine these mappings. There are currently six matchers used by the schema matcher, comprising two schema-level matchers, three element-level matchers and one instance-level matcher.

The two schema-level matchers are the name path matcher and the similar neighbour matcher. The name path matcher constructs paths for the current schema attributes. A path goes from the root of the schema via the relation to the attribute. These paths are then compared using a lexical element-level matcher, which is either the WordNet matcher or the edit distance matcher. A lexical matcher is used because it is more flexible in finding semantic matches than a straightforward string matcher. Options for the name path matcher are excluding the database name and excluding the attribute name from the path. This allows the matcher to analyse only the path to the attribute.

This is useful if the attributes' names are different, but the paths are similar. The second matcher is called the Similar Neighbour matcher, which 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 10% of x by default, but can be adjusted. This matcher runs after all other matchers, using their results to decide which neighbours are affected and by how much.

There are three element-level matchers. The WordNet matcher tokenises the attribute name and then uses WordNet to lookup all synonyms of the base form of each token. This is done for both attributes. Once all synonyms have been found, the similarity is calculated as follows [37]:

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

The edit distance matcher uses the Levenshtein function [15] 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 edit distance matcher will work well for the two attribute names: *FName* and *CustName*. The last element-level matcher is a datatype matcher that uses a datatype compatibility table to specify how compatible two datatypes are, as in the Cupid system.

At the instance-level, a keyword matcher extracts all words from all the stored values of the current attribute. The most frequent words are then compared to a similar list from the other attribute. A similarity value is then calculated using the number of common keywords found. This matcher is particularly useful for stored values that contain text, for example, to detect that *DeptName* is a better match for *Dept* than for *EmpName*. Whilst it can be applied to numerical data, it is unlikely to be as useful.

All these matchers produce their own set of match candidates. Each candidate has a similarity score, which ranges from zero to one. These scores are then combined. The default combination technique used is to calculate the weighted sum. Other combination methods are available and are discussed in the next section. The final match candidates are then sent to the peer for user validation.

Figure 18 summarises the schema matching process. The two input schemas are pre-processed before they are matched. Pre-processing involves tokenising all schema attributes' names, retrieving the base form of those tokens using WordNet, and calculating all the synonyms of those base words, again using WordNet. Other pre-processing is done on the stored values to extract keywords, which are later used by the instance-level matcher.

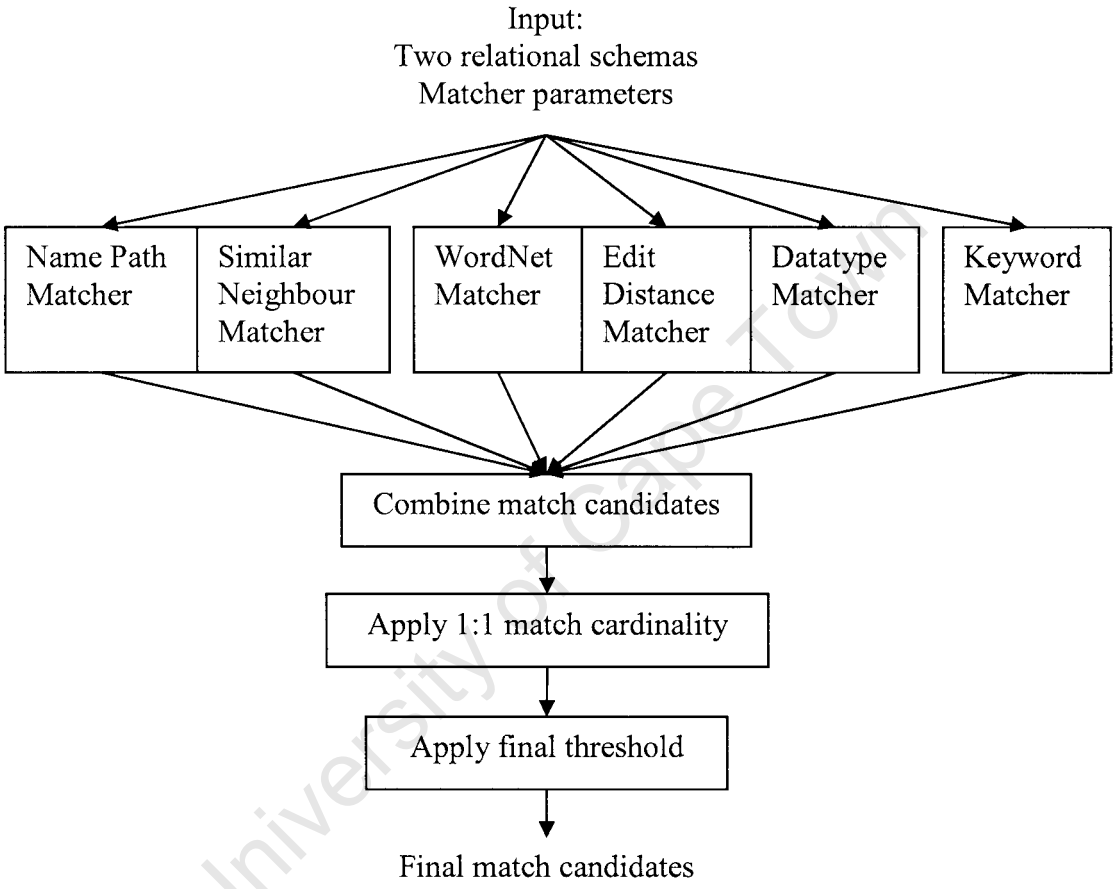


Figure 18: Schema matching process

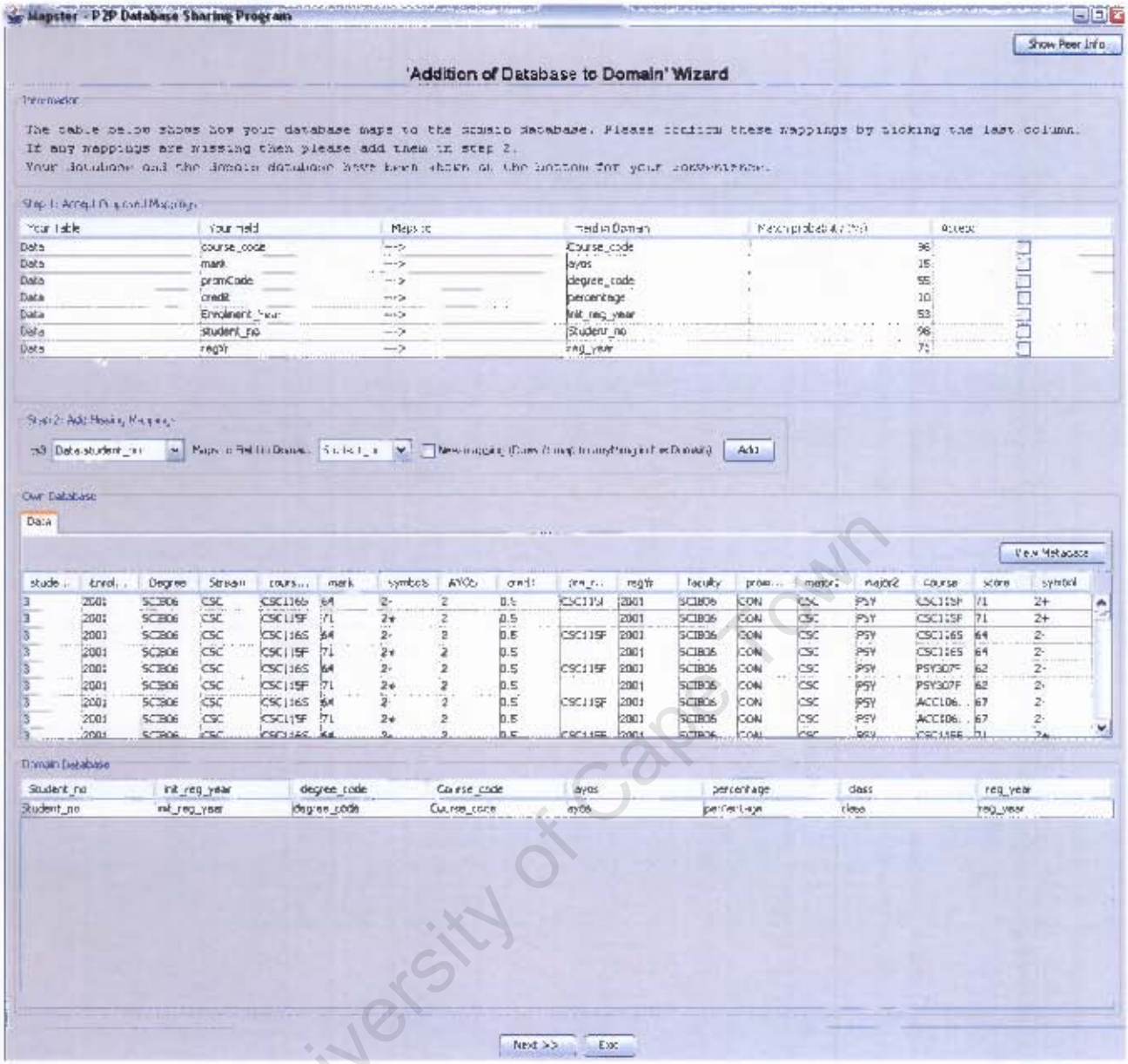


Figure 19: Screenshot of the user validation interface

Figure 19 is a screenshot of the interface that presents the user with the match candidates. This screen is shown after the VSP has processed a user's *join domain* request. The interface shows the user which attributes in the mediated schema have been mapped to his schema. The final match similarity is shown as a guideline as to how likely the system considers this match to be. A check box is provided next to each candidate to allow the user to confirm the matches easily. Both the mediated schema and the user's schema are shown so that the user can refer to them when examining the candidates. The user can also add missing mappings, and can specify if the attribute does not match anything. In this case, the attribute will be put into a new cluster. Once

the user has finished with the match candidates, the matches are checked to see if all attributes from the user's schema have been included. The final matches are subsequently sent to the VSP so that they can be added to the mediated schema.

5.4.1 Schema Matching Test Utility

Schema matching is a complex task and requires several components. This makes it difficult to configure optimally, so a test utility was built. This enabled schema matching to be evaluated before being included in Mapster. It is also useful in its own right as a utility for database administrators should they want to configure the matchers appropriately for their domain. This utility allows a user to test how the matchers perform individually and in combination with others. All parameters for each matcher can be set, in order to improve match accuracy and execution time. Each VSP can use this test package, and the selection of matchers, including their parameters and the way in which they are combined, can then be saved and used in its schema matcher component.

Figure 20 is a screenshot of the interface shown before the match process. A user can specify the input schemas and view them if necessary. He can then choose which matchers to use in the match process. The matchers have been grouped according to the level at which they work. Each section provides the user with information, including suggested default settings, and allows the user to specify parameters specific to the matchers. The global settings allow the user to specify a final threshold, the combination method and whether or not 1:1 match cardinality should be applied, which is useful in reducing the amount of mappings returned to the user. 1:1 match cardinality means that there will only be one mapping for any attribute from either schema. For example, if the attribute *Name* in schema A mapped to *XName* and also *YName* in schema B then the mapping with the lower probability would be removed.

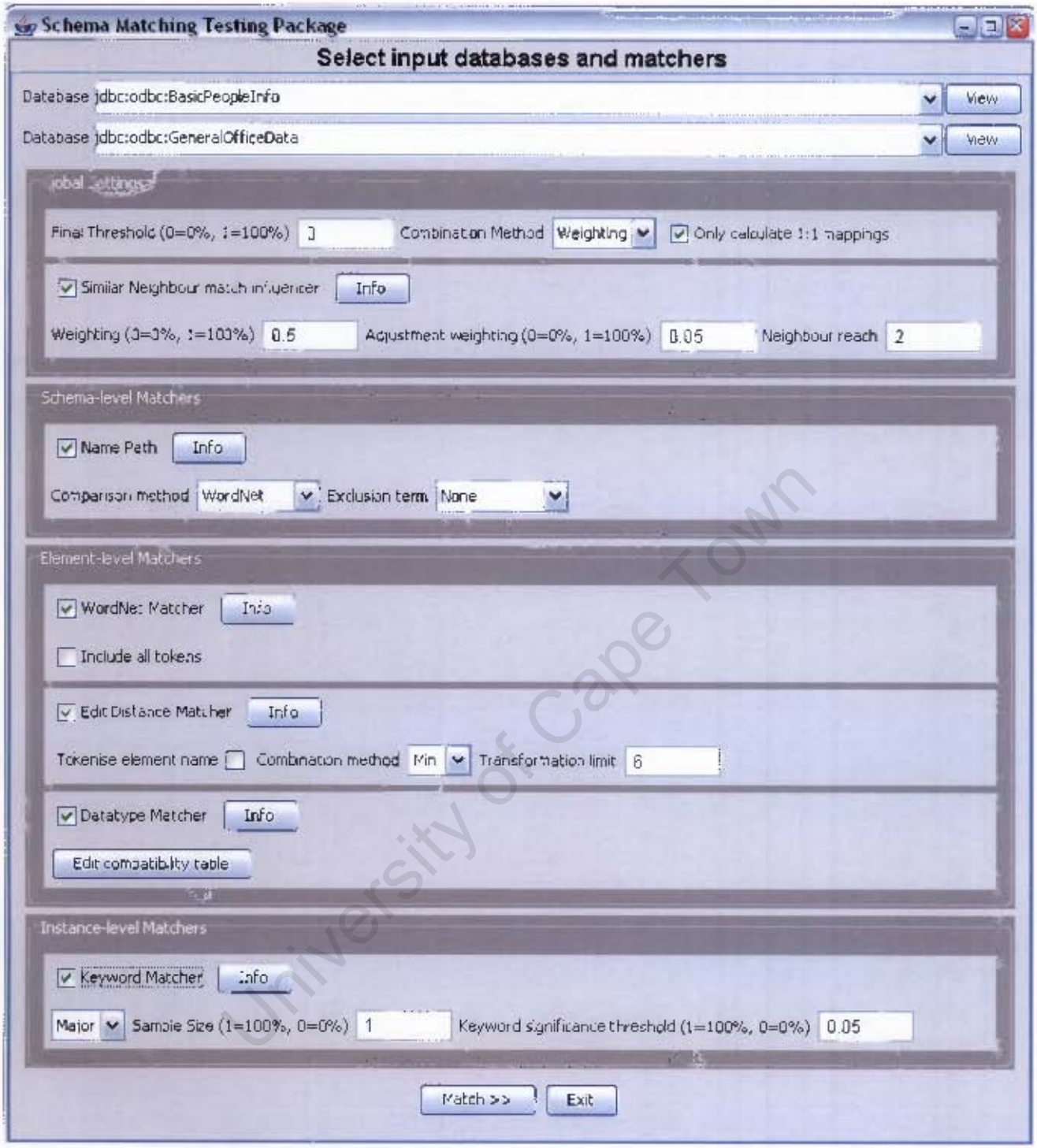


Figure 20: Schema matching test tool: Interface shown before the match process

There are four choices for the combination method, being minimum, maximum, average and weighted sum. Figure 21 is a screenshot of the interface showing the weightings to be used. There is an associated weight for each selected matcher. The default weights are those of the last execution. The weights can be adjusted individually, or equalised using the *Equalise* button at the bottom.

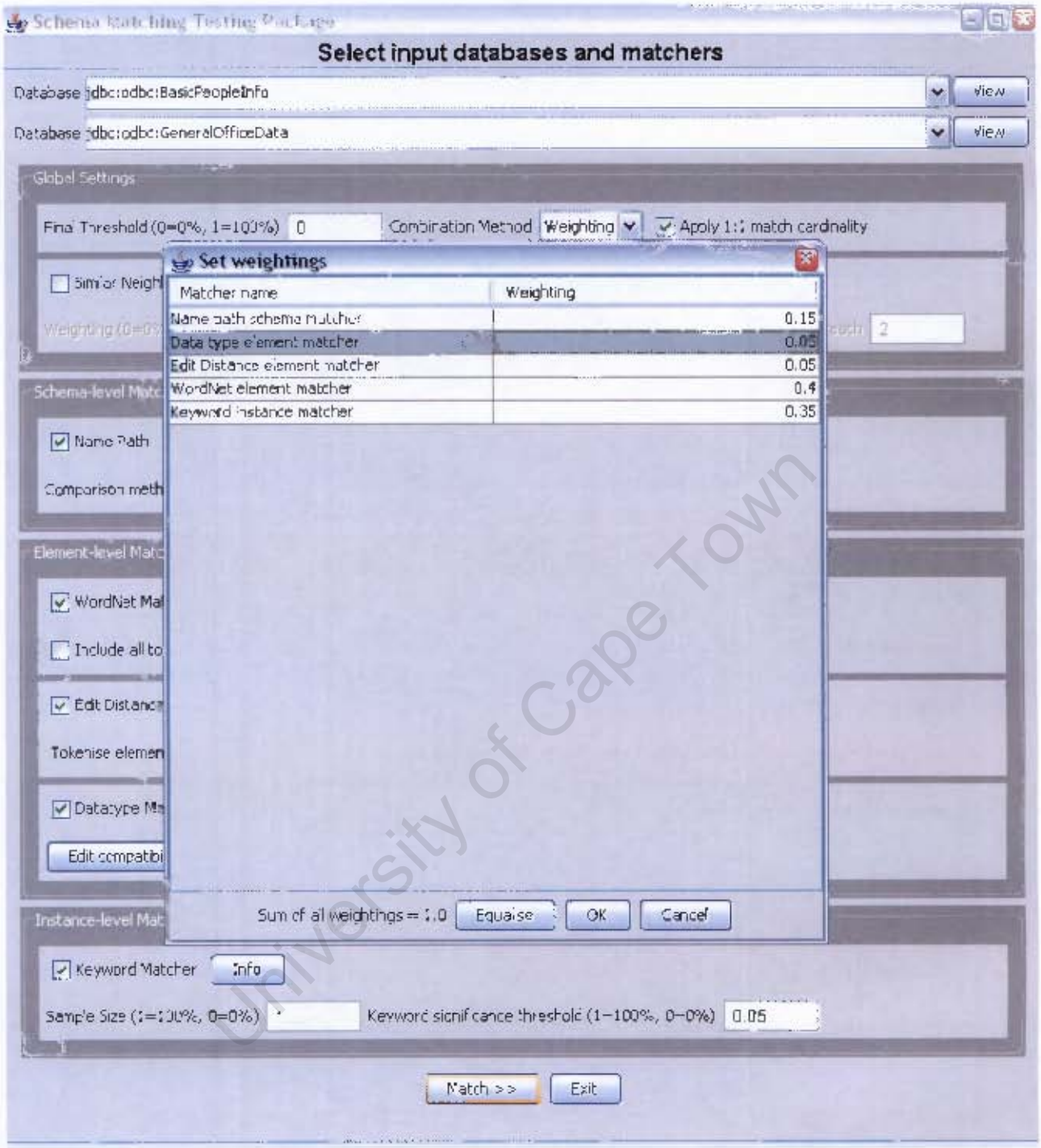


Figure 21: Schema matching test tool: Example of a Weighting Combination

Figure 22 is a screenshot of the interface shown after the match process. The final match candidates are shown at the top. The user can choose to accept them by ticking the check box shown next to each match. The user can add new matches using the combo boxes displayed below the table. When a match is selected, the bottom of the screen

gives a summary of the two attributes and a breakdown of the similarity scores each matcher produced for that pair.

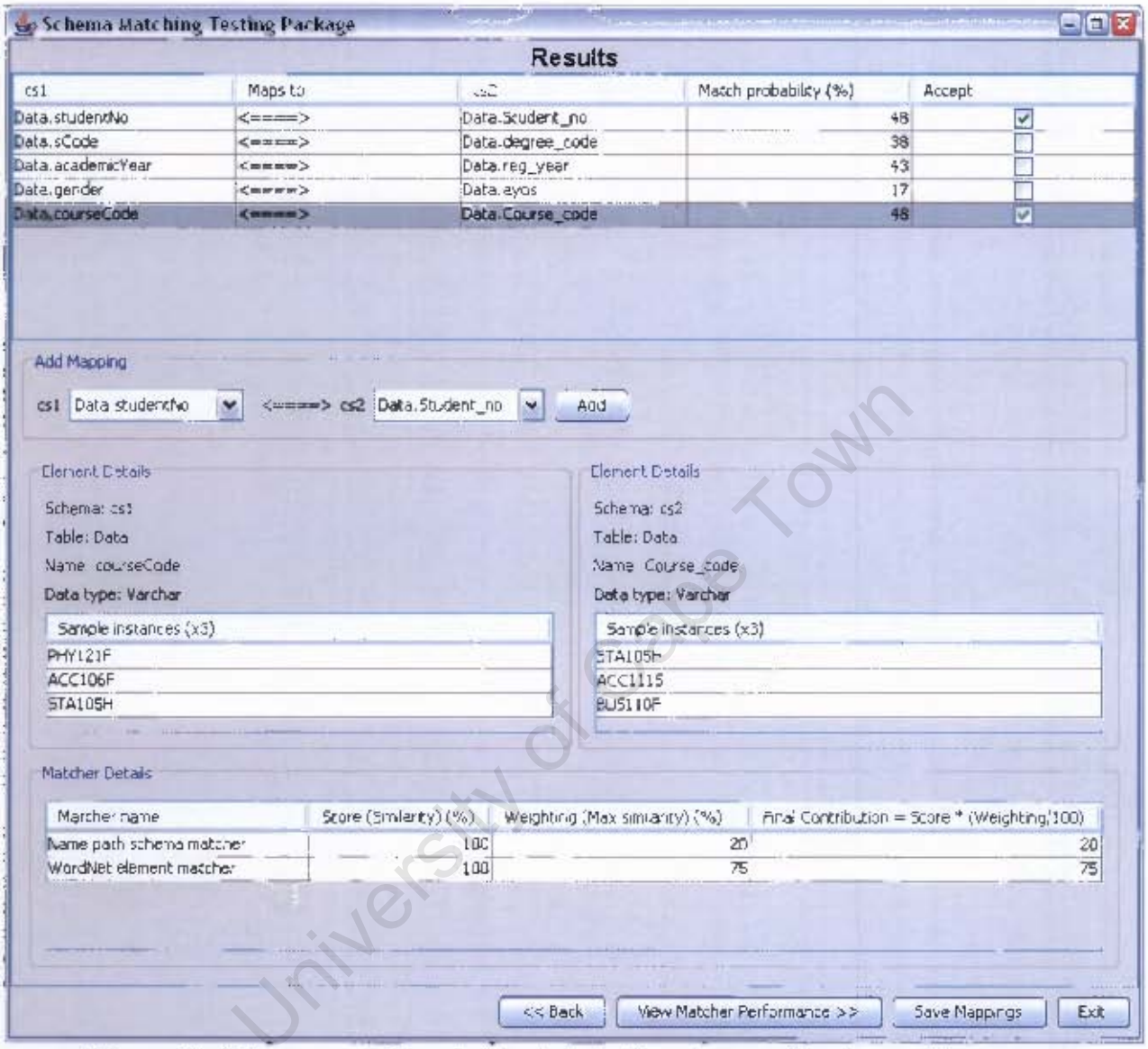


Figure 22: Schema matching test tool: Interface shown after the match process

The final predictions can be evaluated against a set of manually defined matches, which are considered[11] to represent the perfect mappings between the two input schemas. Four evaluation measures are calculated, being Recall, Precision, Overall and F-Measure (see section 3.3 for their definitions). These measures are displayed in a separate dialog box in tabbed form. Each tab corresponds to a matcher's evaluation scores. A combination tab is also provided to see how well the combination of the matchers did.

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

5.5 Query Processor

Figure 23 is a screenshot of the query interface. Queries, which are specified in SQL, are written in the text box. Information and example queries are provided to help the user. The user can specify if the query was written in terms of his schema or in terms of the mediated schema using the check box at the bottom. The user's schema is shown below the text box for reference purposes. The mediated schema can be referred to using the *Mediated Schema* tab.

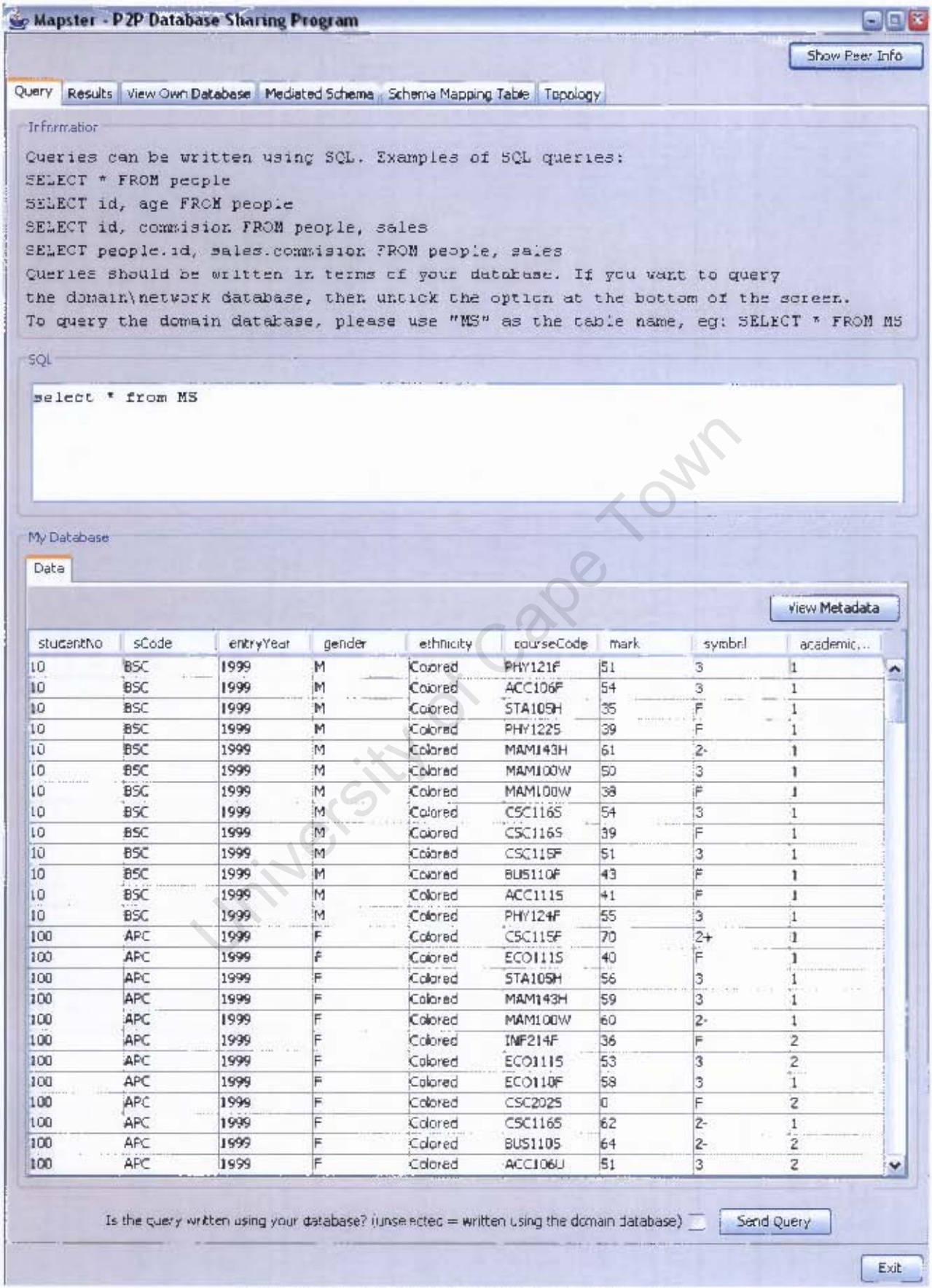


Figure 23: Screenshot of the Query Interface

Once the user has specified the query, it is sent to the peer's ASP. The ASP then processes it so that Peer Queries can be generated. If the query is phrased in terms of the peer's schema, it must first be reformulated in terms of the mediated schema. An example of this process is shown in Figure 24. It involves extracting all attributes from the query. If any attributes are wildcards, then all attributes within the relation are added. The attributes are stored in a hash map where the key is the original attribute in the query and the object is the cluster index in the mediated schema. The attributes in the query are then replaced by these cluster indexes. If the query was written in terms of the mediated schema then the only action is to expand any wildcards.

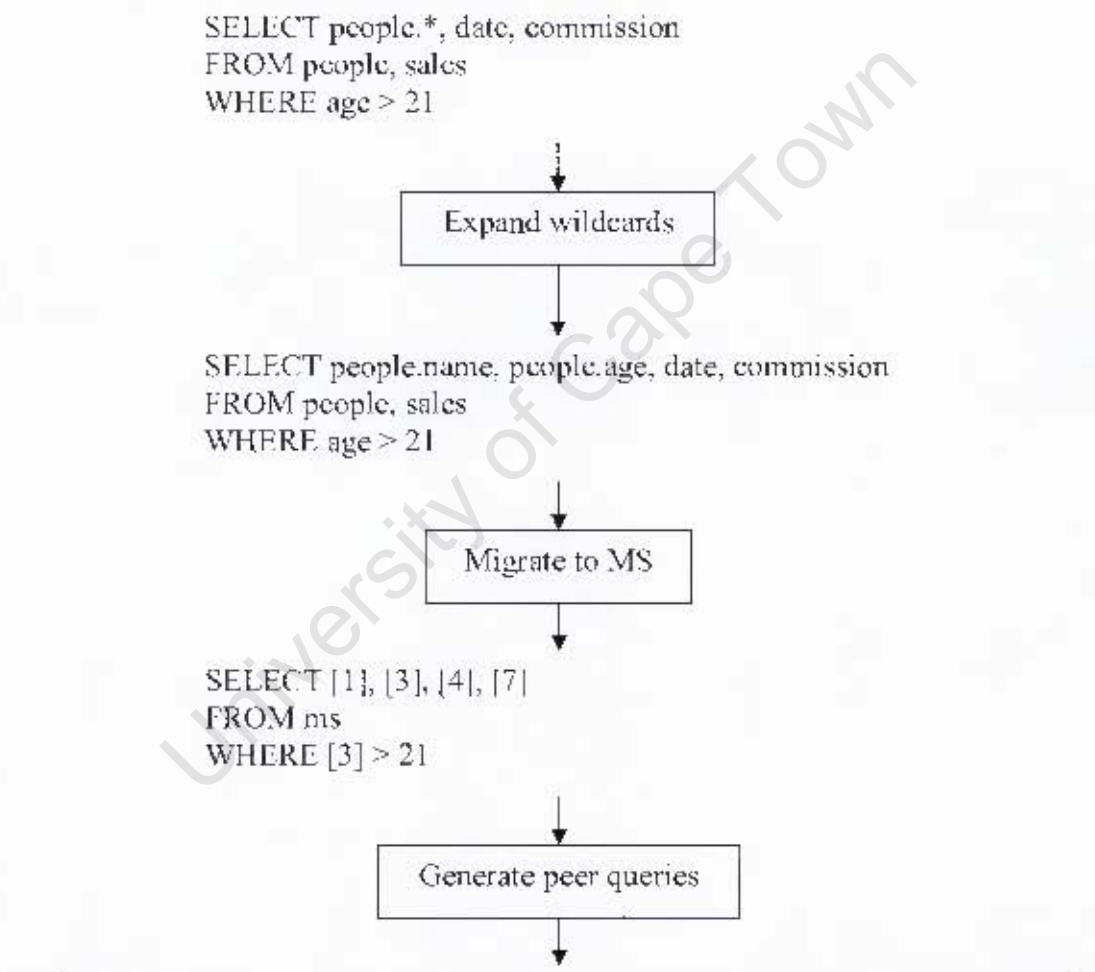


Figure 24: Query processing overview

Peer Queries can now be generated. Cluster indexes from the query are used to find peers that will be involved, i.e. that have schema attributes that will be queried. A Peer Query is then generated for each such peer. The Peer Queries are sent to the relevant peers together with the original peer's ID. The results are sent back directly to the original peer using its ID that was sent with the query.

The original peer waits for results, as each peer has its own workload and is not forced to respond immediately. Results are added to the results interface as they are received. Each set of results from each peer go into a new tab as shown in Figure 25. The results are displayed in a table, along with the generated query that the peer processed. The peer's name is used as the tab name to identify which peers returned which results. If an error occurred then the message is shown where the table would have been displayed. Peers that do not reply with results to their Peer Queries do not affect the working of the current peer. Their results are simply not shown.

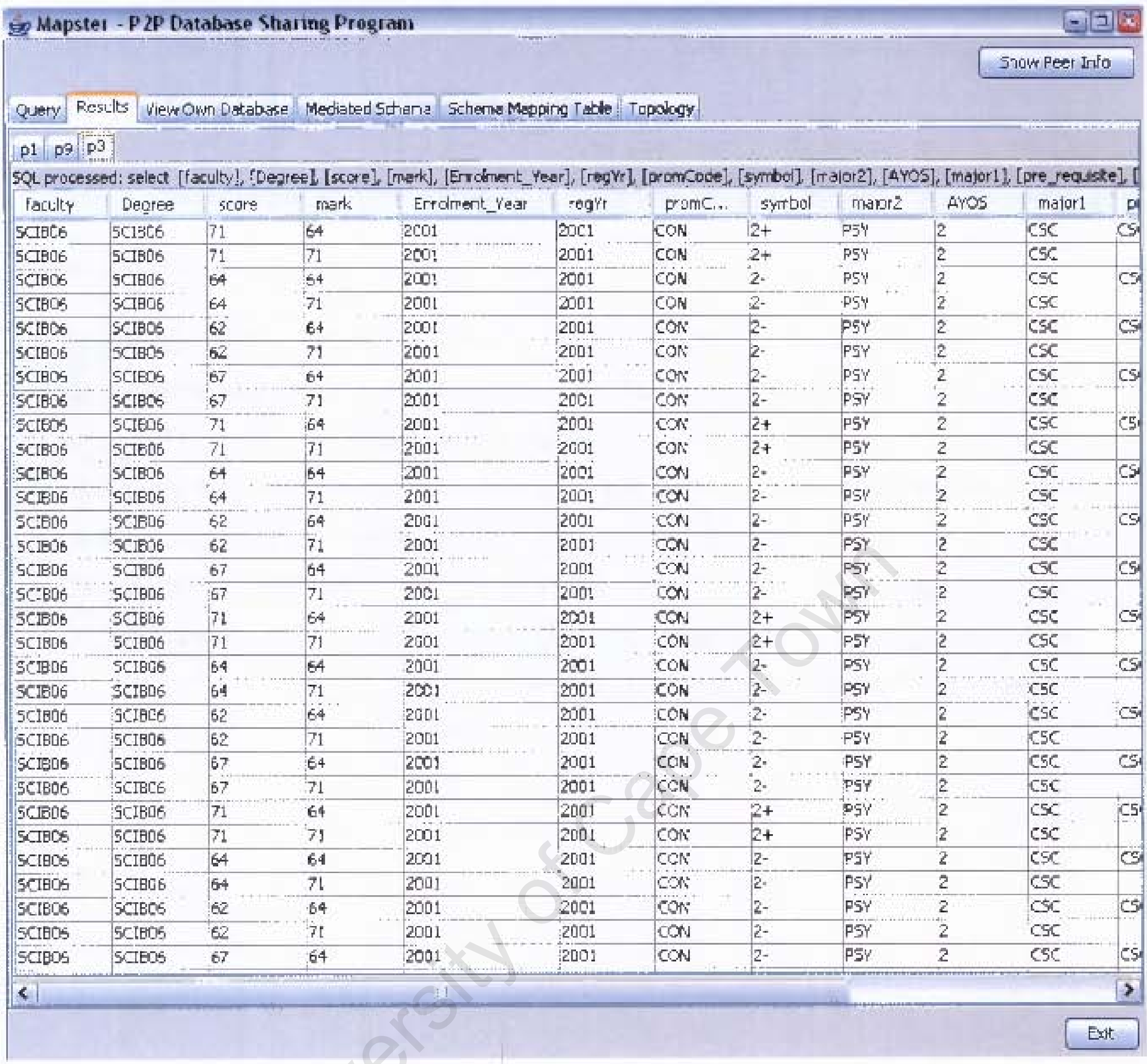


Figure 25: Results interface for the query "SELECT * FROM MS"

As results are received at random, it is inappropriate to combine them into a single table for display purposes. Some schema attributes may not be the same, or even exist, at all peers and combining them may not always make sense. For example, the datatypes may be different or the format of the stored values may be different. Keeping the results separate also gives the ability to show which peers contributed which results. This helps identify which peers are useful.

5.6 Current System

All of the components discussed above have been used in some way to implement Mapster. The fully integrated system works as follows: The first task a peer has is to connect to the network. This requires the user to specify his database and to then log onto the network to obtain a unique ID for the peer. The peer then needs to find a domain to join. Assume for now that this is the only peer in the network, it will have to create a new domain. This means that the peer will become the VSP and that the mediated schema will be the same as the peer's schema. Once the domain is running, the VSP will periodically create and publish a domain advert describing the domain. The VSP is now able to accept new peers and process queries.

Now assume another peer logs onto the network. It will also want to join a domain. By being connected to the network, the peer's JXTA Kernel is able to listen for domain adverts. Let us assume the peer joins the domain created by the first peer. The peer can extract the ID of the domain's VSP from the selected domain advert. It uses the ID to send a *join domain* request, along with its schema[12], to the VSP. The VSP will then use the schema matcher component to match the peer's schema to the mediated schema. The match candidates are then sent back to the peer for user validation. The confirmed matches are sent back to the VSP, where they are added to the mediated schema. The topology manager then decides where to put the peer. Since this is only the first peer to join the domain, it will be added as a child of the VSP. In a larger domain, it would have been added to the most accepting ASP or as an ASP. Note that schema matching and validation occurs only when the peer first joins a domain, as the mappings are saved for future connections.

The above process is illustrated in Figure 26. The sequence of work is shown on the left of the diagram. The text in brackets after each point indicates which component is responsible for that step. JK stands for JXTA kernel, TM means topology manager, MSC is the mediated schema constructor and SM the schema matcher. The reference to GUI in step 6 indicates that the interface plays the major role there.

[12] A sample of the database may be sent later if any instance-level matchers are being used

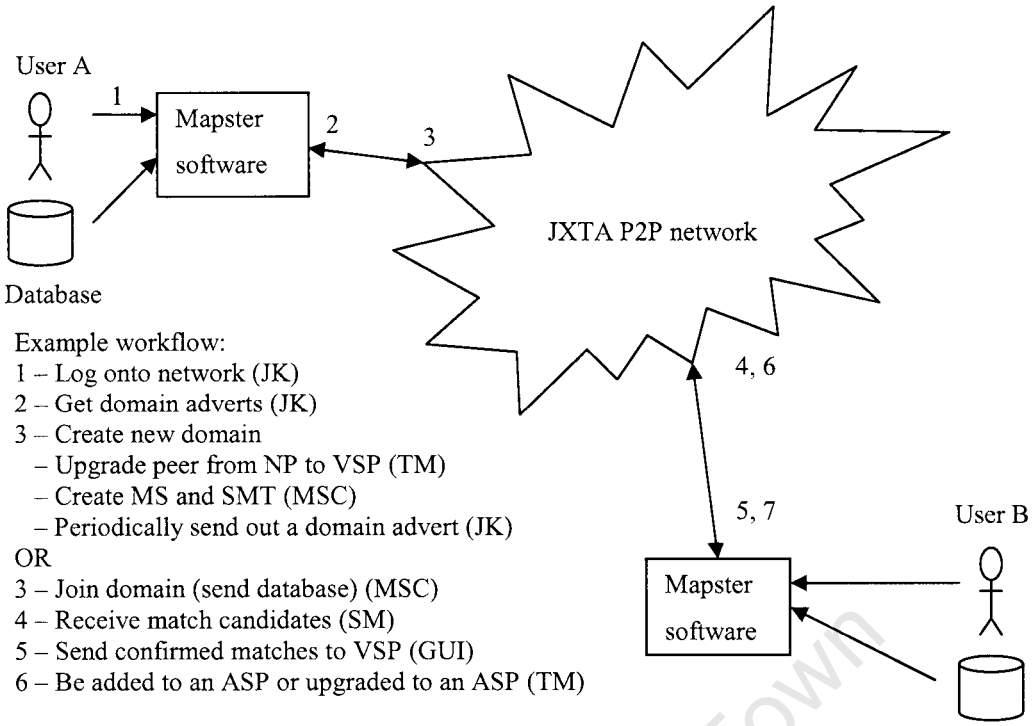


Figure 26: Domain creation and joining a domain for the first time

Chapter 6: Experiments

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. The results are presented in the next chapter. These experiments evaluated the final system; earlier testing was done during development and is not discussed here.

6.1 Schema matching

Thirty-two databases were used in the final schema matching evaluation. These were created independently by students in a third year database course. All databases relate to details about university courses and students. Each database typically had between three and four relations, and an average of 14 attributes. The general similarity between database schemas was about 75%. Since the databases had been created by students for a tutorial and not for real-world use, some of the stored values had to be cleaned and adjusted so that they contained meaningful information. This is highlighted in Table 8 and Table 9. Table 9, unlike Table 8, contains student names that are fake and unlike real-world values. This will not confuse the matchers, but the keyword matcher will not be as accurate as it should be. Since the focus of this research was not purely on schema matching, but rather on its integration in P2P systems, one thorough evaluation based on one set of independently created databases was sufficient to determine if Mapster's matching was good, fair or inadequate.

Table 8: Example relation, called StudentDetails, from one of the test databases

studentNo	name	yearStarted	Res	courseCode	credits
Whtpet001	Peter White	2001	Caravan Park	MAM143H	12
Grygar019	Gareth Grey	2001	Westside	CSC110F	3
Blcfra012	Frank Black	2002	Caravan Park	CSC121S	6
Dlblis078	Lisa Dilbert	2008	Caravan Park	MAM100W	0

Table 9: Example relation, called Data, from another test database

student_no	Name	course	mark	symbol	credits
1	V	MAM111S	23	F	1
2	W	MAM222X	45	FS	3
3	X	CSC100W	66	2+	4
4	Y	CSC100W	78	1	12
5	Z	CSC100W	56	3	7

The schema matching test package was used to perform all evaluations. Each matcher was first configured using six databases in pairs. Each pair was used as input for each matcher resulting in a set of three iterations. Each set was used to test a change in the parameters. Once the best parameters had been chosen, the matchers were run against the remaining 26 databases. This resulted in each matcher being run at least 16 times, 3 times using the default parameters and 13 times using the fine-tuned parameters. Four evaluation measures, being Recall, Precision, F-Measure and Overall, were calculated for each iteration. These measures are explained in section 3.3.

The combination of matchers was then tested. The matchers were combined in one of four possible ways and run against a set of six databases. The combination was done by taking the minimum, maximum, average or weighted sum. These combination methods are explained in section 5.4. The best combination was then run against all databases to calculate average evaluation measures. In total, about 140 iterations were done.

6.2 P2P Architecture

To test the P2P system, a trace facility was added to record data that related to several tasks, including logins, reconnections and querying. Login time was measured from when the peer selected a domain to when it could start to query that domain. Reconnection time was the time taken for an online peer to reconnect to the domain when its parent or the VSP went offline. Query time was split into two categories, being simple queries and complex queries. Simple queries only contained "select from" clauses, i.e. attribute projections, and complex queries were all remaining ones. The recorded data included completion times for the given tasks and domain information, such as the size of the mediated schema, and the super-peer and normal peer count.

Updated domain information was propagated down from the VSP to the ASPs using the ping managers. Every time the VSP received a ping from an ASP, it replied with the current domain statistics. The ASPs in turn forwarded the information onto its peers in a similar fashion. Each peer saved these times in its own XML file.

A settings file for each peer controlled the testing so that human interaction was not required and would therefore not affect the results. This included bypassing message boxes and handling the login process in code. SP willingness values were also set in the file so that specific executions of the network could be tested. Schema matching was automated using the best parameters obtained from testing the schema matcher. The testing system changed the network randomly, with different peers coming and going, and different peers becoming the VSP, etc.

Testing the P2P network was tedious and only a selected number of tests were run from many possible tests. Due to limited resource availability, testing was done on four machines, each running between three and five instances of Mapster, on a LAN, and no tests were run longer than two hours. This resulted in a network of 17 peers each using a different database, which were the same that were used in the schema matching tests. The network ran at 100Mbps and the machines all had P4 processors with approximately CPU speeds of 1.6GHz, and 256Mb RAM. Each test was run five times.

6.3 Usability

Mapster and the schema matching test package were also tested for usability. During the design and implementation stages, walkthroughs and workflows [27] were drawn up to visualize how a user would use the system. These provided a basis around which the system could be created. Once implemented in the JXTA database sharing prototype and the schema matching test package, a pilot study [35] was done to test how effective these interfaces were and to streamline future testing and development. This study highlighted problems in the schema matching test package. The focus of the test package had been to provide the user with access to all the matchers and the necessary parameters. This was changed to make the interface more intuitive to a first time user. Several interface improvements were added as a result of the pilot study.

After the final implementation, usability tests were conducted, including direct observation [48] and questionnaires. After the experiment, which involved performing several tasks using Mapster and the schema matching utility, each user was given a questionnaire to fill in. Twelve users were involved, with the first half being computer scientists and the second half being users with average computer ability. The first half was asked to test both the schema matching utility and Mapster, whilst the second half only tested Mapster, since the schema matching test package is aimed at database and system administrators.

Testing the schema matching utility was done by asking the users to select any two databases from a collection of available databases and then to match them. Matching was performed three times. The user first used a predefined combination of matchers, so that the ease of use of the user interface could be tested. The second task asked the user to choose their own combination of matchers. This tested how well the users understood the capability of the package from its interface. The last task asked the users to refine the parameters for the matchers that they chose in task two. This tested how well the interface supported this task and if the information, presented with each matcher, was sufficient.

The testing of Mapster was more involved. It required the users to perform several tasks and fill out a questionnaire. The questionnaire has been included as Appendix 1. The questionnaire was drawn up using guidelines from [27] and explained what the test was about, what Mapster is, and some of the concepts used in the system. It also contained the tasks the users had to perform. Tasks ranged from joining the domain to simply querying the domain. Joining the domain involved selecting an appropriate domain and then confirming the mappings that were determined by the VSP. The user's database had been chosen beforehand to ensure that the user had to adjust some mappings, confirm some, and add some new mappings. This tested all aspects of the *Join Domain* interface. The task descriptions were kept very brief, such as "join the University domain", so that the user had to figure out the steps. This tested how intuitive the interface was to use. Feedback was gathered during the execution of the tasks using direct observation [48]. The questionnaire, completed after the tasks, asked the user to consider areas that the tasks did not cover, such as if there was any functionality that

was missing, what improvements could be made, etc. The last question, which asked the user to give an application of the system, tested how well they understood the system and its purpose.

Chapter 7: Findings

This chapter covers the findings of the experiments described in the previous chapter. The first section examines the results of the testing done on the schema matching component, including how well the individual matchers performed and the combination thereof. The results of testing the P2P architecture is then presented in section 7.2, followed by the results of the usability study.

7.1 Schema matching

The schema matching utility produces mappings with m:n cardinality. This means that the system may produce several mappings from an attribute in database A to different attributes in database B. For example, *Name* in database A may be mapped to both *Fullname* and *CName* in database B. This can be useful in providing the user with a choice of mappings, but can also confuse the user by providing too many options. It therefore improves the Recall value, but at the expensive of Precision; see section 3.3 for an explanation of evaluation measures. Therefore, only mappings with 1:1 cardinality were produced like other systems [25, 40, 22, 59], for the final tests, so that the best results from the matchers were evaluated.

The performances of the individual matchers are now discussed, followed by the combination of matchers. Scores used in all figures are the result of 16 tests of the 32 databases. Each figure displays the minimum, average and the maximum scores observed for each evaluation measure. The minimum score is the worst observed value in all tests, whilst the maximum is the best observed. The average is the average evaluation score calculated from all tests.

7.1.1 Schema-Level Matchers

Name Path Matcher

This matcher compared paths of attributes by using one of two lexical matchers, being the WordNet matcher or the edit distance matcher. In tests, the WordNet matcher performed better than the edit distance matcher did. The edit distance matcher worked well when 1:1 match cardinality was not applied, but returned too many candidates when 1:1 match cardinality was applied, which lowered the evaluation measures,

especially Overall. Exclusion terms could be specified, such as excluding the schema name or the attribute name. The initial testing showed that no term should be excluded. The results from 16 tests on the 32 university databases are shown in Figure 27. The matcher achieved an average Overall score of 0.22 and an average Precision of 0.64.

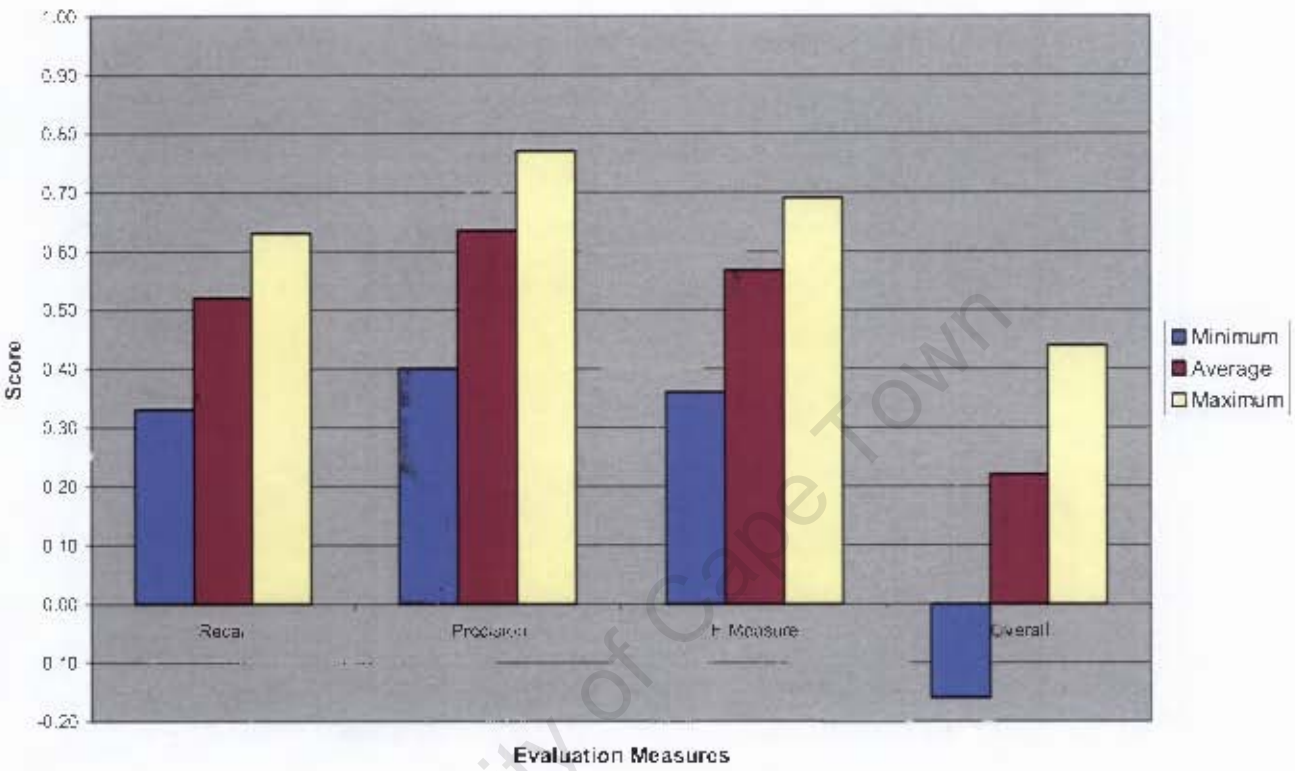


Figure 27: Evaluation measures for name path matcher

These results are not as high as they should be, because of the relation names that were used. For example, some databases had relations with names such as *Table1* and *temp_t*. Exploiting relational constraints, such as keys, would help, as better and more detailed paths could be created and used in the analysis.

Similar Neighbour Matcher

This matcher was difficult to test, as it uses the results of other matchers as input to its own match process. When results were obtained, they alternated between returning far too many results or none at all. This was caused by the databases lacking composite attributes, such as *Address*, which is usually made up of several attributes, including *street number*, *street name*, *suburb*, etc. Match similarities would propagate amongst all the attributes and all similarities would increase, creating new matches that were not

justified. This matcher would be useful with more complex schemas, which would be more decomposed. Similarities would then be correctly propagated and only the appropriate attributes would be involved. This matcher was not used in the final combination of matchers.

7.1.2 Element-Level Matchers

WordNet Matcher

This was the second best matcher. The only parameter that can be set is the option of including all tokens in the comparison, including non-English tokens that cannot be used by the WordNet dictionary. Initial tests showed that all tokens should be included. The results in Figure 28 show that the matcher performs consistently well. The average Overall score was 0.29 and average Precision 0.7.

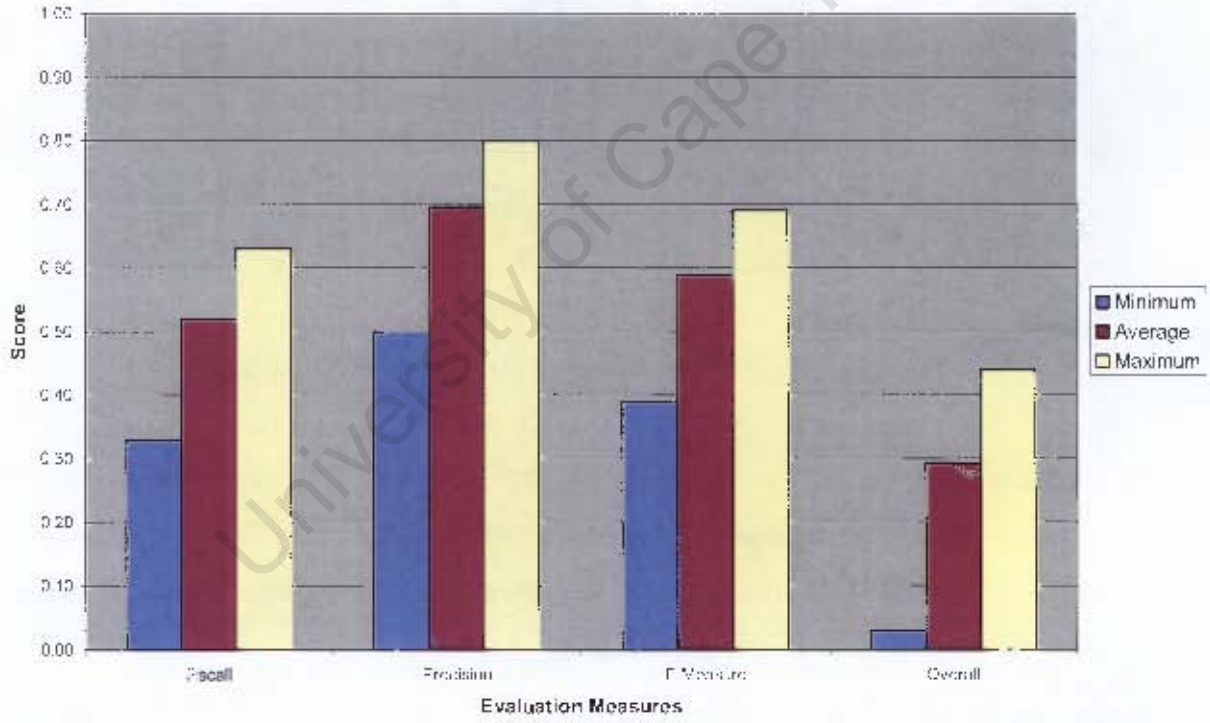


Figure 28: WordNet Matcher Evaluation Scores

Many systems use a name matcher, as the attribute name should be the primary indicator of the meaning of the attribute. This makes it an essential matcher to get correct. The WordNet dictionary was useful, but should be complemented with a collection of acronyms, abbreviations, and suffixes and prefixes. WordNet cannot

handle such words and these were quite common in the test schemas. The matcher would perform far better if all words could be analysed, therefore the additional source would be useful.

Edit Distance Matcher

The edit distance matcher performed poorly, as shown in Figure 29, where the minimum Overall was -0.25. The only parameter that could be set was to specify if the attribute name should be tokenised. This was shown to perform worse than just comparing the whole attribute name and was therefore not used. The matcher scored an average Overall of 0.06 and an average Precision of 0.54.

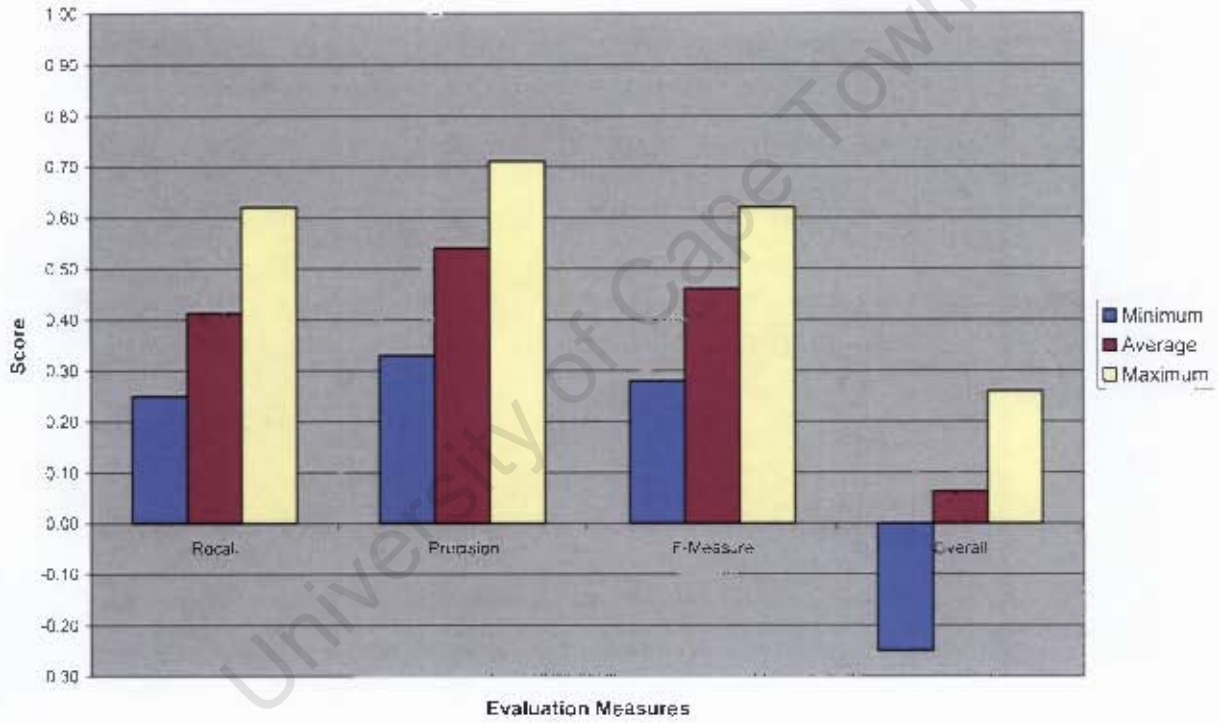


Figure 29: Edit Distance Matcher Evaluation Scores

The summary table at the end of chapter 3 shows that only two systems have used this matcher and that the more recent systems have not used it. The poor performance experienced here explains why this matcher has not become popular.

Datatype Matcher

The datatype matcher required lots of pre-match effort to set up the datatype compatibility table. Each SQL datatype was assigned a compatibility value to every

other type that was considered compatible to it. This resulted in over 40 entries. During the pre-match setup of the databases, it was evident that the database creators did not care about datatypes as most attributes were simply specified as textual attributes. While it is likely that datatypes might be better specified in a real-world database where data validity is crucial, in practice this matcher is not very applicable, as a datatype often fails to reflect the actual type of data. The results in Figure 30 illustrate how little this matcher contributed. The minimum Overall was -0.12, whilst the average was -0.03. The negative average indicates that it would be more work to correct the mappings this matcher produced than to determine them manually. The average Precision was 0.33.

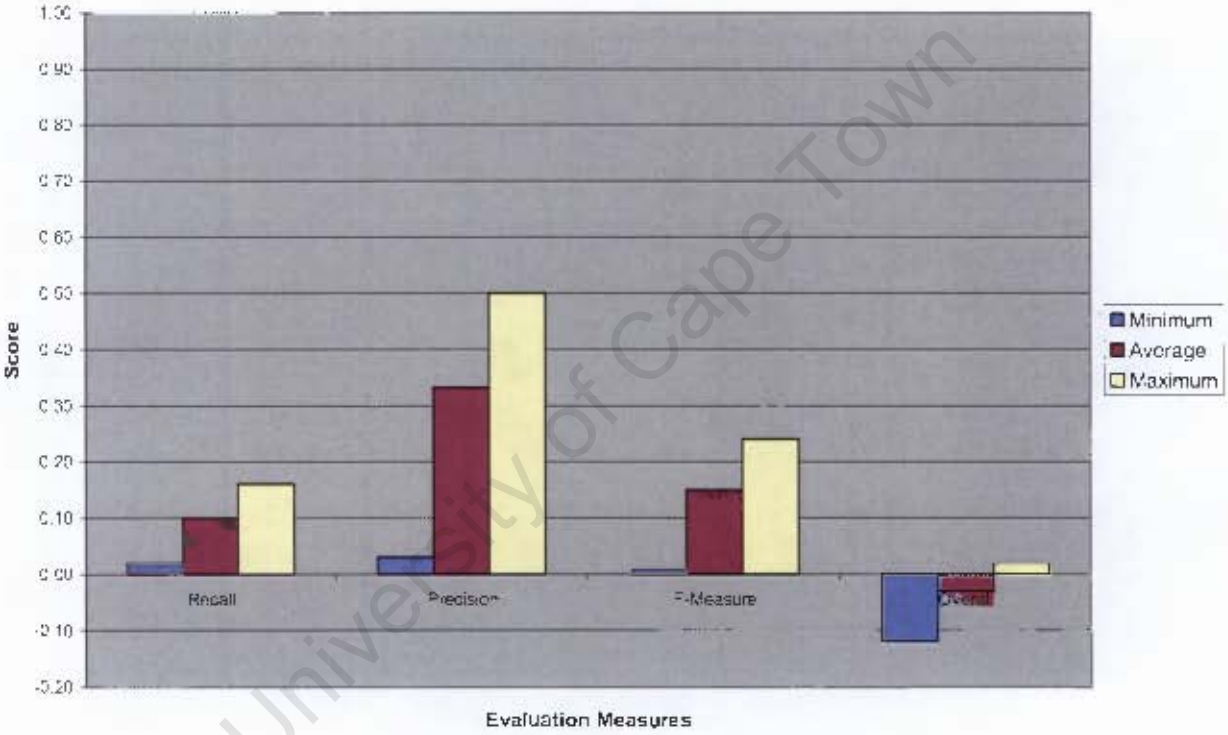


Figure 30: Datatype matcher evaluation scores

Previous systems [59, 40] have found that the datatype matcher should have a low weighting towards the final match score. This is confirmed by these results.

7.1.3 Instance-Level Matcher

Keyword Matcher

This matcher performed surprising well, considering how small the students' databases were. The values in Figure 31 show that this was the best matcher, closely followed by

the WordNet matcher. The main parameter defines the threshold for what is considered a keyword and what is considered noise. The number of times a token appeared was divided by the total number of tokens to calculate the frequency of this token from the attribute's stored values. If this frequency was higher than the given threshold then it was considered a keyword. This threshold was set at 5%. Initially it was set at 25% but this was too high and too many keywords were excluded from the comparison. The maximum Overall was 0.83, whilst the average Overall was 0.34. The average Precision was 0.73.

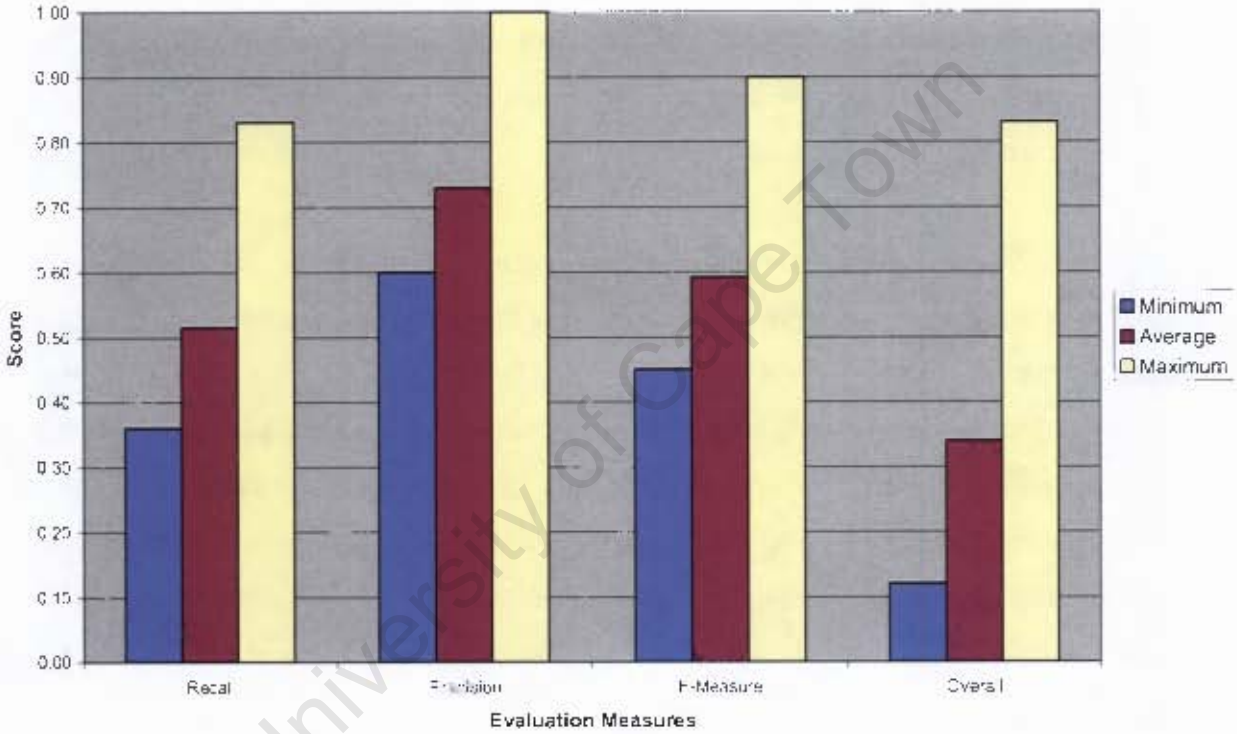


Figure 31: Keyword matcher evaluation scores

The good performance of this matcher highlights how much information is available in the stored values and that schema matching systems should focus on exploiting them more. Future systems should look to add at least one type of instance-level matcher. Machine learning matchers should be focused on as most exploit past matches and they often perform well. If an instance-level matcher is used with other matchers then the instance-level matcher should be weighted more than the other matchers should.

7.1.4 Combined Matchers

The matchers could be combined by calculating the average or the weighted sum, or by taking the minimum or maximum, of the scores from all the matchers. The last two options performed poorly in initial tests and were not included in further tests. Calculating the average is a special case of calculating the weighted sum, where the weights are all equal.

Figure 32 illustrates three combination methods that were tested. The scores shown are averages from 16 tests. Weighting 1 used the following weights: 50% for the keyword matcher, 30% for the WordNet matcher, 10% for the name path matcher and 5% for both the datatype matcher and edit distance matchers. These weights were based on the performance of the individual matchers. The keyword matcher was weighted heavily, whilst weaker matchers, such as the edit distance matcher, were given very low weights.

Weighting 1 performed worse than the average option. The results of Weighting 1 were used to create a new weighting called Weighting 2, which used the following weights: 45% for the keyword matcher, 45% for the WordNet matcher, and 5% for the name path matcher. This combination did away with unnecessary matchers and increased the weighting of the WordNet matcher. This was done to allow the system to handle attributes that had stored values that were not similar, even though the attributes were semantically equivalent. Weighting 2 performed the best with an average Overall of 0.41 and an average Precision of 0.71. The various evaluation measures for Weighting 2 are shown in Figure 33.

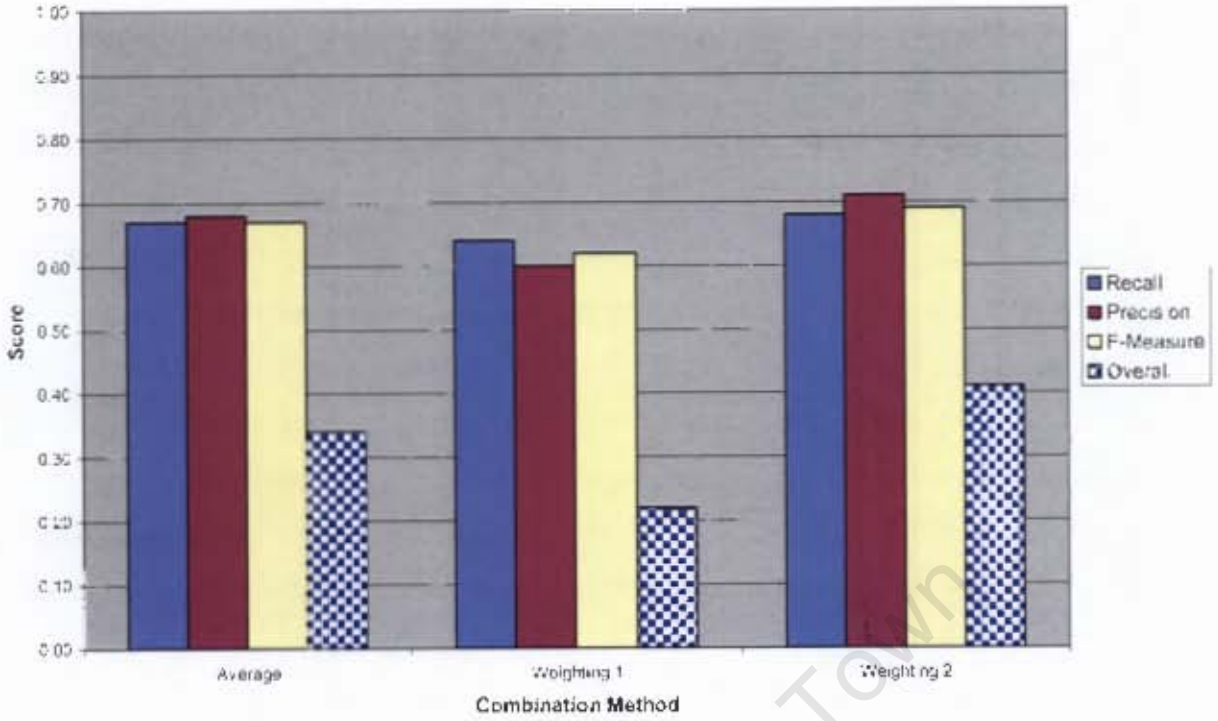


Figure 32: Comparison of Different Combination Methods

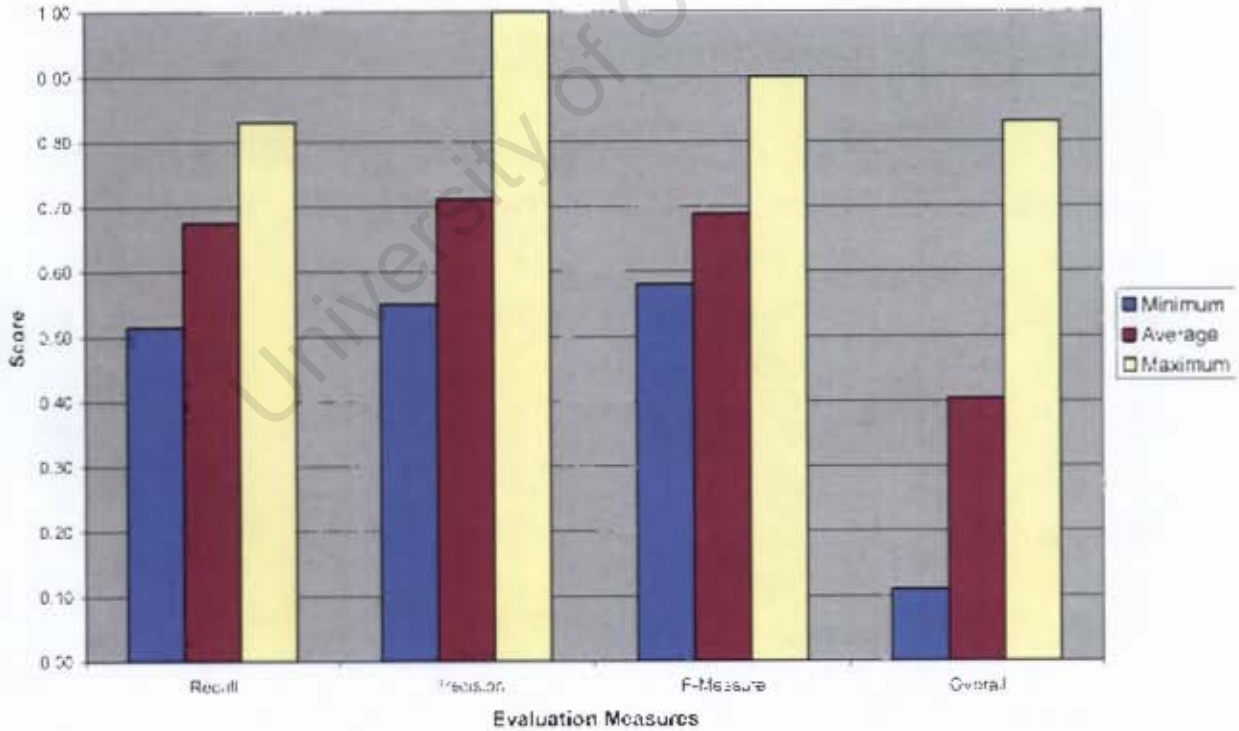


Figure 33: Evaluation Scores for the Combination of Matchers

Average Precision shows that the schema matcher correctly detected over 70% of the matchers automatically, hence saving the user a considerable amount of work. The

average Overall value of 0.4 is lower than other recent schema matching systems [23, 22, 40], which score between 0.6 and 0.8. The best Overall score was 0.82, which was also the Overall achieved by COMA. Unfortunately, only four of the eleven systems described in chapter 3 have tested their systems using one or more of the schema matching evaluation measures. In addition, the systems use different test schemas. This makes it difficult to compare scores and performance. However, recent schema matching systems are dedicated to improving schema matching and employ some very sophisticated techniques to determine the matches. Therefore, this system did not do too badly considering that the work explores the use of schema matching in a P2P system, and hence studies typical representative matching techniques rather than attempting to be complete or break new ground in schema matching per se. We conclude from the experiments that the schema matcher performed more than adequately and served its purpose as a representative of semi-automated schema matching.

7.1.5 Time

Although fast schema matching was not a goal of this work, times were recorded to compare matchers accordingly. Very few systems report execution time and so it is not well known how fast existing schema matchers are.

Figure 34 illustrates how fast the matchers used in Mapster were. The test databases were small, typically containing about 15 attributes and about 1200 stored values; nevertheless these times are relatively fast. The keyword matcher is slow, as it must analyse all stored values. The name path matcher is slow, as it must construct the schema path for each schema attribute. The combination is considerably slower than individual matchers are. This can be improved by running each matcher in its own thread, particularly as matchers often need to wait for data to be read from disk. The edit distance and datatype matchers ran very quickly.

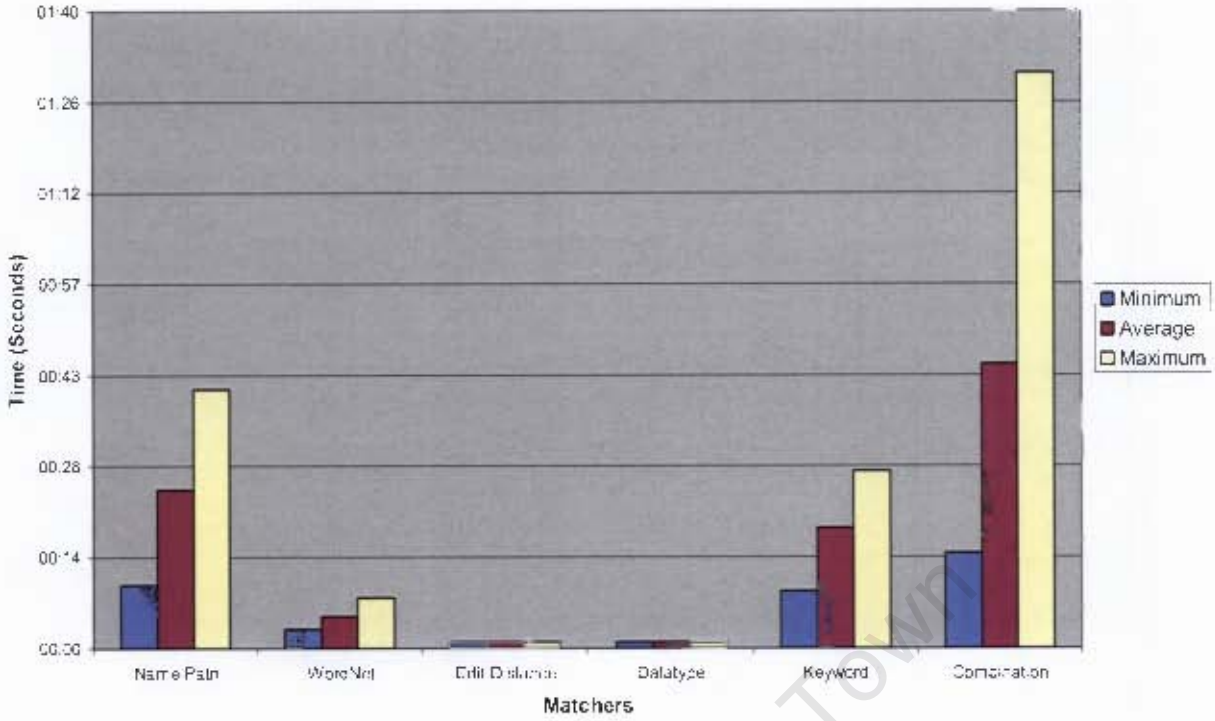


Figure 34: Execution Times of the Different Matcher Configurations

7.2 P2P Architecture

Figure 35 illustrates how the mediated schema within a domain grew as the number of peers connected to that domain, increased. Each peer's database contained about 14 attributes and the 17 peers had a combined total of over 220 attributes. The figure illustrates how well the mediated schema scales as it only contained 26 clusters after all 17 databases had connected.

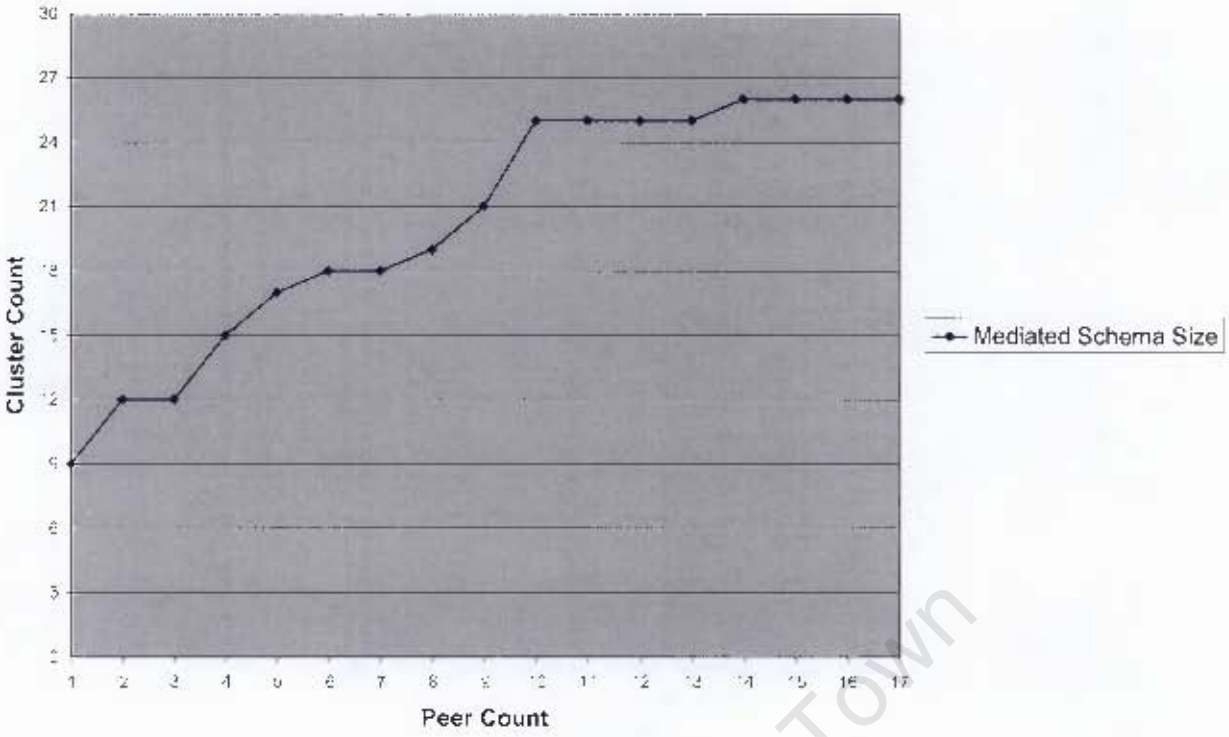


Figure 35: Mediated Schema vs. Peer Count

Although the testing of the schema matchers was done using the schema matching utility, it was noticed when using the schema matcher in the P2P network that the quality of the proposed mappings improved noticeably. The chance that a new attribute matched the appropriate cluster improved as the number of attributes within the clusters grew.

It was found that the times used by the ping managers needed tweaking. The time used by the normal peers to ping their ASPs was too frequent and was changed from 15 to 40 seconds. This was due for two reasons. They were initially set too low, but as the number of peers within a domain increased, it was evident the super-peers required more time. A time of 40 seconds should suffice for large domains.

Testing the P2P network examined how fast tasks were performed. The three main tasks are logins, querying and reconnecting.

7.2.1 Logins

Login time is the time taken from when a peer selects a domain to when he can query it. Peers joining a domain for the first time are not timed, so that the time taken to perform schema matching did not affect the test. The login process was fast, with the average being just over three seconds and the maximum being under 10 seconds. Figure 36 shows the logins times for a peer count of 17. The graph starts with a peer count of two as the first peer creates the domain and does not login to it.

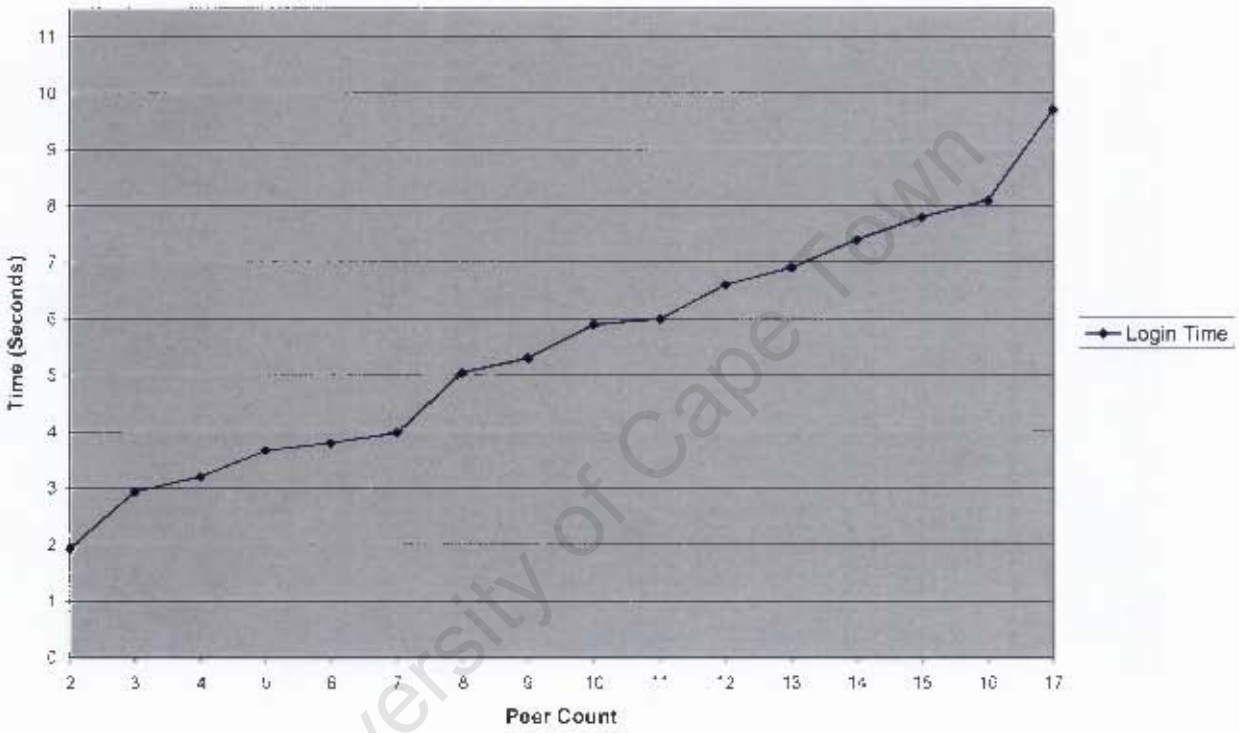


Figure 36: Login Time vs. Peer Count

The login time appears to be linear. The constant increase is because the VSP must match the new schema to an increasingly higher number of attributes in the mediated schema. The VSP must then determine the best place for the peer within the domain according to the topology manager. This may involve upgrading the peer to a super-peer, which can be noticed in the graph. The time taken for the 17th peer to join is longer as it must change into a super-peer. This is caused by the value defining how accepting super-peers were of new peers, called NP acceptance. This value was set at four for all peers.

7.2.2 Querying

Query time is measured from when the peer sends the query to when it receives the results from all peers that were involved in the query. Figure 37 illustrates the time taken for each type of query. There is not much difference between the query types, indicating that the execution times do not depend on the type of query but rather the peers and the network. In addition, complex queries are more specific than simple queries and so return smaller data sets.

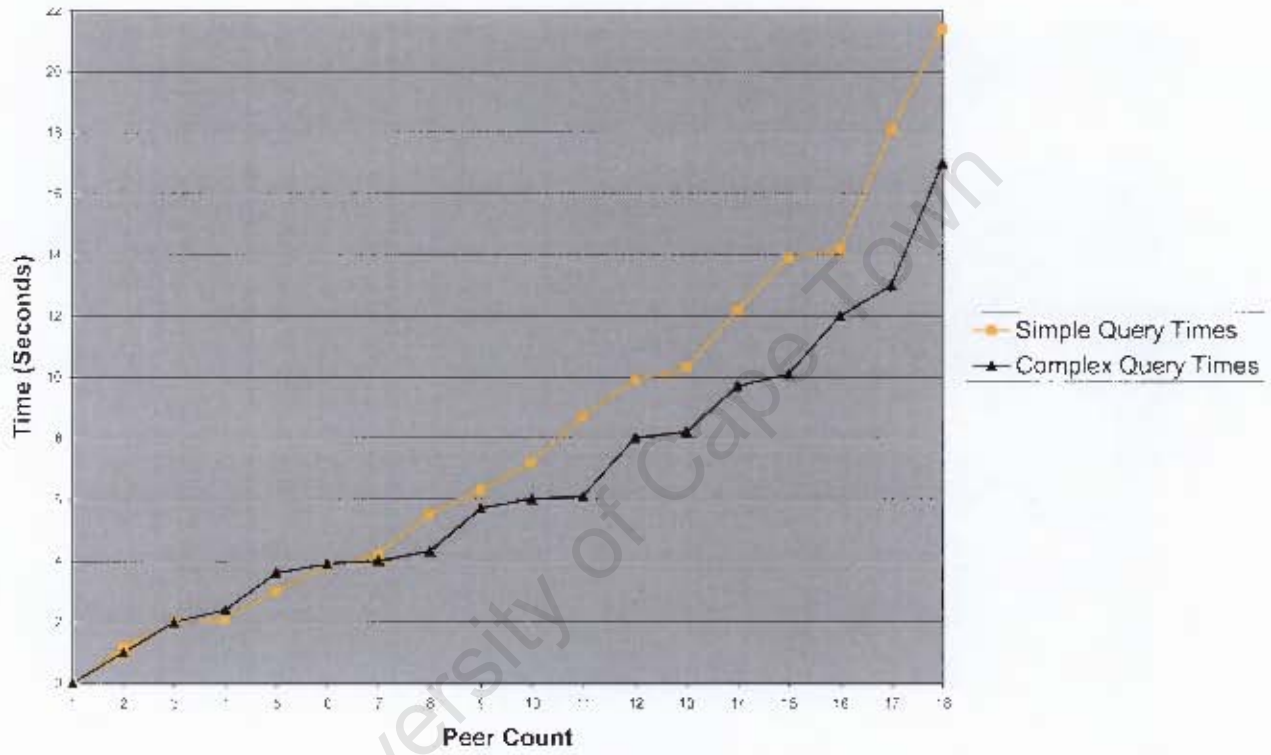


Figure 37: Query Times vs. Peer Count

Figure 38 illustrates the difference in query times between the types of peers that initiate the query. Super-peers get the results far sooner than normal peers do. On average, normal peers wait nearly twice as long for all results. This difference is caused by the fact that the super-peers have a copy of the mediated schema with them. A normal peer does not have a copy and has to send the query to its parent super-peer.

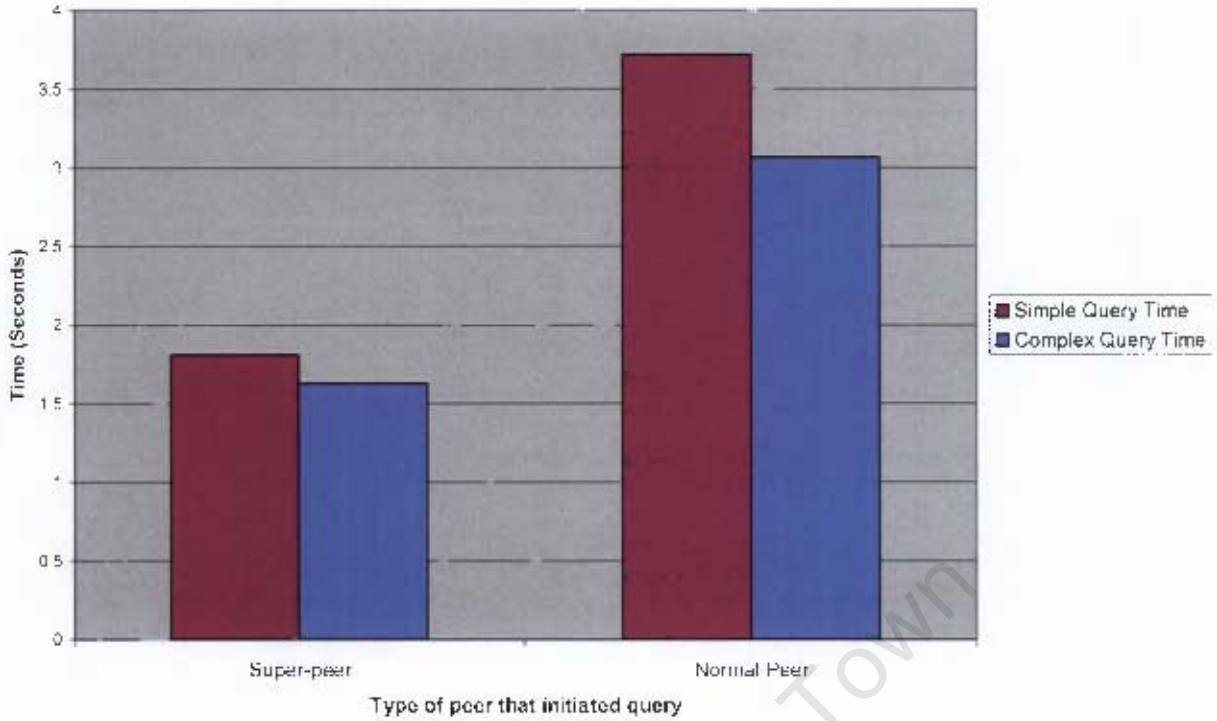


Figure 38: Query times vs. type of peer that initiated query

7.2.3 Reconnecting

When a super-peer goes offline, all its children must reconnect. Either a normal peer's parent goes offline or the VSP goes offline. If a parent super-peer goes down then the peer just reconnects. If the VSP goes down then the process takes at least 10 seconds, as all the ASPs must first decide on a new VSP and all ASPs must then connect to the new VSP.

Figure 39 displays the time taken for peers to reconnect to a domain when their parent super-peer goes offline. The times are faster than login times, as the peer only needs to ask the VSP for a new location within the domain. The peer's schema is already mapped to the mediated schema and every ASP already has a copy of all mappings in the domain.

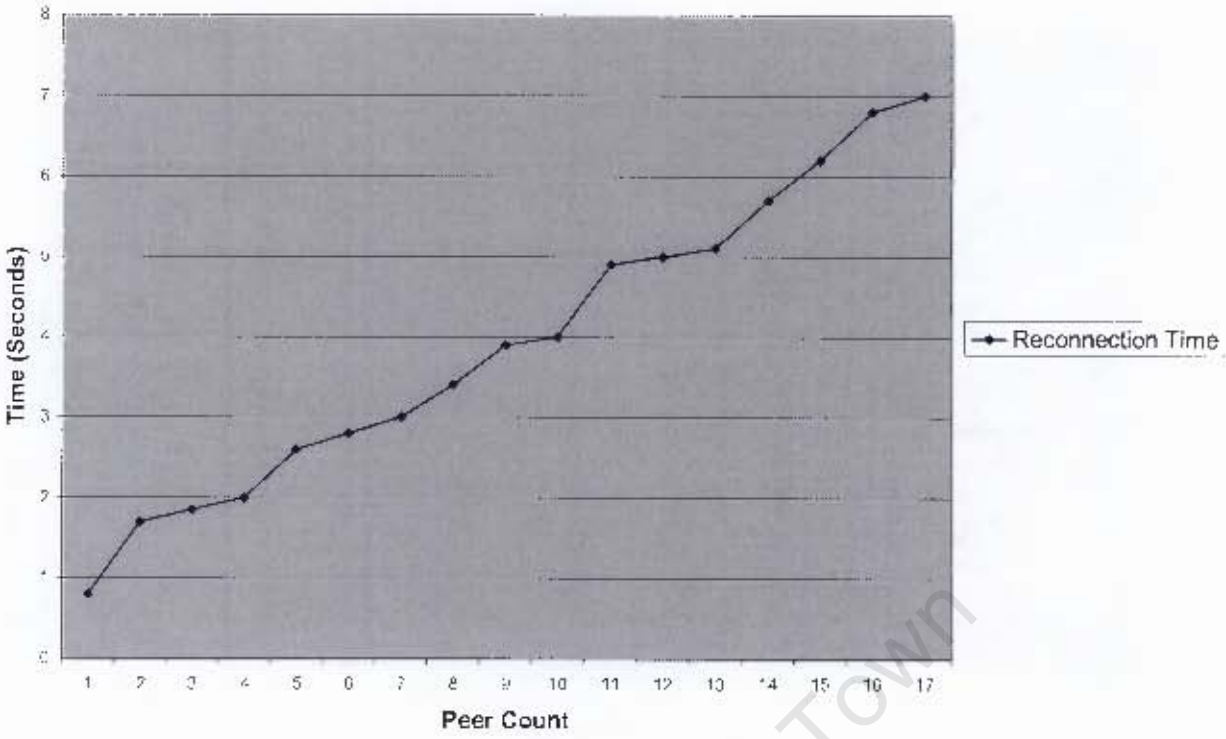


Figure 39: Reconnection Time vs. Peer Count

The success of these results is difficult to determine. It is unclear what typical performance is for a P2P database-sharing system. Other P2P database-sharing systems either have not performed similar tests or have not published results.

7.3 Usability

Five of the six users found the schema matching test utility easy to use. The tasks were completed easily and all users understood what the interface allowed a user to do. Only three users read about each matcher, in the information panels, before using them. The others simply fiddled with the values until the results improved. One suggested that the interface used to display the determined mappings be replaced by an interface that shows the two database's schemas and lines linking the determined matches. This would easily show which attributes had been mapped and to where they mapped. New mappings could then be dragged from one schema to the other. This interface would require a lot of work to implement and it is unclear whether all users would prefer it, so it was left for future investigation. Another user suggested using colour to denote how similar attributes were. For example, red would indicate matches with scores below 35%, yellow could be for 35% to 65% and green for anything above 65%. However, the

table containing the results can be sorted according to score and so this colour was not needed. In addition, adding colour would add another item that the user would have to remember when checking the mappings. This change was omitted to keep the interface simple.

Feedback on Mapster resulted in several changes being subsequently incorporated, but most changes were simple, like changing captions to be more user friendly. For instance, the mediated schema was renamed the domain database. More information panels were added, especially in the query interface, and the domain joining process used a wizard that followed a step-by-step procedure. Three of the twelve users found the domain joining interface tiresome to use. They found it tedious to check mappings and add new ones for all attributes in their schema. The users did find it very useful to have their database shown alongside the mediated schema when adjusting the mappings produced by the schema matcher. The mediated schema was displayed using a table, where a cluster was shown as a column. The rows made up the attributes that mapped to the various clusters. It may be beneficial to include stored values in this display as well. It is uncertain how to achieve that. When told about the idea to have a graphical model where mappings could be created by dragging attributes from schema A to schema B (discussed in the previous paragraph), seven users thought that would be easier to use than the current interface.

Finally, five of the non-technical users did not find the query interface easy to use. They found it tedious to write out the SQL queries. They said that they would have preferred an interface similar to the query-by-example interface used in Microsoft Access [60]. That interface allows a user to drag attributes from the schema on to a table, which represents the final query. Some constraints can be added to the table by specifying them below the attribute in the table. On the other hand, some SQL queries cannot be written using this method and require text input. A future implementation should have the options of the query-by-example or text input to cater for all users and types of queries.

During the experiments, it was observed that eight of the twelve users did not read the information and help displayed at various stages in the application. They found the

summary given in the questionnaire and the interface sufficient to allow them to figure out what to do. Users also found subsequent use of Mapster very straightforward, particularly the fact that they only needed to adjust the mappings between their schema and the mediated schema once.

7.4 Summary

The schema matcher performed well and was sufficient for use in Mapster. Adding another instance-level matcher would help improve that component. The use of super-peers and a mediated schema within a domain made the schema matching process efficient and robust. Times captured during testing show that the P2P architecture scales well, at least for the number of peers used. The usability testing showed that the schema matching utility was easy to use, whilst Mapster required a few minor changes, which have since been implemented. The mapping interface used in the schema matching test utility and the domain joining interface could be changed in future to use a graphical model that allows the user to link attributes by dragging them around.

Chapter 8: Conclusion

This chapter summarises the work presented in this thesis and suggests some avenues for future research.

8.1 Summary

A few P2P database-sharing systems now exist [3, 29, 41, 42]. This research explored aspects of P2P database-sharing systems that need attention, specifically better mechanisms for coping with schema differences. Matching of attributes across different database schemas should be automated as much as possible. The user interaction component, required to confirm or alter the mappings produced by the schema matching process, should provide an easily usable interface, which explains how matches were determined. The maintenance of schema matches as peers join and leave should be efficient and effective, as should the use of this information when querying remote databases.

This research first investigated P2P database sharing systems and schema matching mechanisms. Mapster was built to provide a P2P environment that is suitable for existing schema matching techniques. The system uses a super-peer 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 super-peers using the schema matching techniques. The construction of the mediated schema can be done as the peers join the network, which allows matching do be done before a query is posed, thus speeding up query processing. Clustering according to the peers' area of interest should ensure that the shared schemas contain overlapping data, which has been shown to be very useful in improving schema matching accuracy [7]. Testing showed that the approach is viable and that the system's usability, schema matching ability and performance are very promising.

Previous systems have used either pairwise mappings, which do not scale well, or global mediated schemas, which can be impractical if there are many databases and/or database overlap is low. Hence, Mapster has domains or areas of interest with their own mediated schemas, and uses pairwise mappings across domains. The use of mediated

schemas allows queries to be answered far more easily and quickly than is the case when pairwise mappings are done. As far as I can ascertain, Mapster is the combines mediated schemas with pairwise mappings and to focus on the network topology to enable effective semi-automated schema matching. It is also the only P2P database-sharing system to provide a range of configurable schema matching techniques and a graphical user interface, which gives details of the matches identified and facilitates manually altering these. Super-peers make mediated schema maintenance scalable, which is important in P2P networks where there can be any number of peers present at any time.

8.2 Future work

The following section covers some of the more important concepts that should be researched in future.

8.2.1 Additional Schema Matchers

Mapster provides a complete system that performs schema matching and database sharing. It does not focus solely on schema matching. The schema matching component lacks a powerful instance-level matcher and the ability to learn from past matches. Bayesian networks [40] are the preferred machine learning approach to analysing stored values within a database [25]. When stored values are in raw form, this may take too long. An alternative is to convert the stored values into patterns, whereby, for example, the letter *n* would replace numbers, and strings would be replaced by the letter *s*. For example, the entry *crouse@cs.uct.ac.za* would be converted into the pattern *s@s.s.s.s*. This will allow for faster processing without much loss of precision.

Several recent systems [18, 25, 37] have begun to focus on past matches and exploit this information to determine new matches. Not only can new matches be determined or learnt, but new acronyms, jargon and abbreviations can be compiled from previous mappings. One such approach could be to create a collection of acronyms using past mappings and combine it with a web service. One such web service could come from www.AcronymFinder.com, which contains a vast collection of acronyms and abbreviations. However, one of its developers said that they are unwilling to make this information available as a web service, for now [39].

8.2.2 Interface for Schema Matching

More research needs to be done on how best to present mappings to users, either for viewing or for adjusting. Using graphical links to represent mappings between two schemas is one possibility. If these links could be manipulated to change mappings and to add new ones then it might ease post-match effort required by the user. Clio [67] is a system that focuses on mapping creation through the user interface and has several features that could be useful in addressing this issue.

8.2.3 Indexing and Caching

Performance, especially for querying, was better whenever a super-peer initiated the query rather than a normal peer. This was because the super-peers had a copy of the mediated schema with them. If the mediated schema was cached in more places then query processing would be faster. However, copies would age and require updating. Indexing could also be used to direct queries to the "best" peers, i.e. peers with low workload or better resources. More research needs to be done into these options.

8.2.4 Case study

The system has been tested through experiments using schemas from several domains and networks of varying size and behaviour. A case study in which the system is tested by real users in a real-world application is needed to determine how effective Mapster is in practice.

8.2.5 Inter-domain querying

For inter-domain querying, a VSP needs to listen continuously for domain adverts. Every time a domain advert is received, the list of known domains must be updated. Some kind of cooperation between ASPs could enable them to share such lists. A peer could use the ID from the domain advert to contact the relevant VSP, which would send the mediated schema back to the peer.

8.3 Conclusion

This thesis has achieved its goal of building a viable system for semi-automated schema matching in P2P database-sharing systems. Querying databases in such an environment appears to be an exciting and worthwhile avenue for future work, with the long-term benefits potentially having significant impact on electronic data sharing.

Bibliography

[1] Adar, E., and Huberman, B., (2000), Free on Gnutella, First Monday: Peer Reviewed Journal on the Internet, URL: http://firstmonday.org/issues/issue5_10/adar/index.html, last accessed: 17 May 2005.

[2] Androutsellis-Theotokis, S., and D Spinellis, D., (2004), A survey of peer-to-peer content distribution techonologies, ACM Computing Survey 36(12) pp335–371.

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

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

[5] Bawa, M., Cooper, B., Crespo, A., Daswani, N., Ganesan, P., Garcia-Molina, H., Kamvar, S., Marti, S., Schlosser, M., Sun, Q., Vinograd, P., and Yang, B., (2003), Peer-to-Peer Research at Stanford, Computer Science Department, Stanford University, Stanford, CA, USA.

[6] Bender, M., Michel, S., Triantafillou, P., Weikum, G., and Zimmer, C., (2005) MINERVA: Collaborative P2P Search, in Proceedings of the 31st VLDB Conference, Trondheim, Norway.

[7] Bergamaschi, S., Castano, S., Vincini, M., and Beneventano, D., (2001), Semantic integration of heterogeneous information sources, Data and Knowledge Engineering 36: 3, pp215–249.

[8] Berlin, J., and Motro, A., (2002), Database Schema Matching Using Machine Learning with Feature Selection, Information and Software Engineering Department, George Mason University, Fairfax, VA, USA.

[9] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I., (2002), Data Management for Peer-to-Peer Computing: A

Vision. Technical Report, Microsoft Research, Microsoft Corporation.

[10] Boncz, P., and Treijtel, C., (2004), AmbientDB: Relational Query Processing in a P2P Network, Amsterdam, The Netherlands.

[11] Botros, S., and Waterhouse, S., (2001), Search in JXTA and Other Distributed Networks, Sun Microsystems, Inc., Palo Alto, CA, USA.

[12] Castano, S., De Antonellis, V., Fugini, M., and Pernici, B., (1998), Conceptual Schema Analysis: Techniques and Applications, ACM Trans. Database Systems 23: 3, 286-333.

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

[14] Clifton, C., Housman, E., and Rosenthal, A., (1997), Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases, The MITRE Corporation, Bedford, MA, USA.

[15] Cohen, W., Ravikumar, P., and Fienberg, S., (2003), A Comparison of String Distance Metrics for Name-Matching Tasks, Centre for Automated Learning and Discovery, Carnegie Mellon University, Pittsburgh PA, USA.

[16] Crespo, C., and Garcia-Molina, H. (2001), Routing Indices for Peer-to-Peer Systems, Stanford University, Stanford, CA, USA.

[17] Daswani, N., Garcia-Molina, H., and Yang, B., (2002), Open Problems in Data-Sharing Peer-to-Peer Systems, Stanford University, Stanford, CA, USA.

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

[19] Dictionary.com, URL: http://www.dictionary.com, last accessed: 14 January 2006.

[20] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B., (2003), Globally Distributed Content Delivery (IEEE Internet Computing, October 2003).

[21] Do, H., Melnik, S., and Rahm, E., (2002), Comparison of Schema Matching

Evaluations. Technical report, University of Leipzig, Leipzig, Germany.

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

[23] Doan, A., (2004), Schema and Ontology Matching: Current Research Directions, Database and Information System Group, University of Illinois, Urbana Champaign, USA.

[24] Doan, A., Domingos, P., and Halevy, A., (2001), Learning to Match the Schemas of Data Sources: A Multistrategy Approach, University of Washington, Seattle, WA, USA.

[25] Doan, A., Domingos, P., and Halevy, A., (2001), Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach, ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA.

[26] Gong, L., (2002), Project JXTA: A Technology Overview, Sun Microsystems, Inc., Palo Alto, CA, USA.

[27] Granollers, T., and Lorés, J., (2005), Cognitive Walkthrough With Users: An alternative dimension for usability methods, University of Lleida, GRIHO - Avda. Jaume II, 69 – 25001.

[28] Gribble, S., Halevy, A., Ives, Z., Rodrig, M., and Suciu, D. (2001), What Can Databases Do for Peer-to-Peer? In Proceedings of the 4th International Workshop on the Web and Databases (WebDB), Santa Barbara, May.

[29] Halevy, A., Ives, Z., Mork, P., and Tatarinov, I. (2003), Piazza: Data Management Infrastructure for Semantic Web Applications (WWW2003, May 2003, Budapest, Hungary).

[30] Halevy, A., Ives, Z., Mork, P., and Tatarinov, I., (2003), Piazza: Data Management Infrastructure for Semantic Web Applications, University of Washington, Seattle, WA, USA.

[31] Halevy, A., Ives, Z., Mork, P., and Tatarinov, I., (2003), Piazza: Data Management Infrastructure for Semantic Web Applications, University of

Washington, Seattle, WA, USA.

[32] He, B., and Chang, K., (2003), Statistical Schema Matching across Web Query Interfaces, in SIGMOD 2003, June 9-12, 2003, San Diego, CA, USA.

[33] He, H., Meng, W., Yu, C., and Wu, Z., (2003), WISE-Integrator: An Automatic Integrator of Web Search Interfaces for E-Commerce, in VLDB 2003, Berlin, Germany.

[34] Kang, J., and Naughton, J., (2003), On Schema Matching with Opaque Column Names and Data Values (SIGMOD 2003, June 2003, San Diego, CA, USA).

[35] Kasunic, M., (2004), Conducting Effective Pilot Studies, Carnegie Mellon

Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA.

[36] Kementsietsidis, A., Arenas, M., and Miller, R., (2003), Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues (SIGMOD 2003, June 2003, San Diego, CA, USA).

[37] Madhavan, J., Bernstein, P., Chen, K., Halevy, A., and Shenoy, P., (2003), Corpus-based Schema Matching, University of Washington, Seattle, WA, USA.

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

[39] Malloy, M., personal communication, August 2005.

[40] Melnik, S., Garcia-Molina, H. and Rahm, E., (2002), 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.

[41] Ng, W. S., Ooi, B. C., and Tan, K. L., (2002), BestPeer - A Self-Configurable P2P System. Technical report, School of Computing, National University of Singapore.

[42] 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.

[43] Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., and Loser, A., (2003), Super-Peer Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks (WWW2003, May 20–24, 2003, Budapest, Hungary).

[44] Ooi, B., Shu, Y., and Tan, KL., (2001), Relational Data Sharing in Peer-based Data Management Systems, Department of Computer Science, National University of Singapore, Singapore.

[45] Palopoli, L., Terracina, G., and Ursino, D., (2000), The System DIKE: Towards the Semi-Automatic, Synthesis of Cooperative Information Systems and Data Warehouses, ADBIS-DASFAA 2000, pp108–117.

[46] Pant, G., Srinivasan, P., and Menczer, F., (2002), Exploration versus Exploitation in Topic Driven Crawlers, Department of Management Sciences, School of Library and Information Science, The University of Iowa, Iowa City, IA, USA.

[47] Perry, P., (2001), Scalable P2P Search, URL: http://www.paulperry.net/notes/p2p.asp, last accessed: 9 April 2005.

[48] Plaisant, C., Rose, A., Shneiderman, B., and Vanniamparampil, A., (1996), User interface reengineering: Low effort, high payoff strategies, IEEE Software, vol.14, 4 (July/August 1997) 66-72.

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

[50] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., (2001), A Scalable Content-Addressable Network (SIGCOMM'01, August 2001, San Diego, California, USA).

[51] RDF Primer: W3C Recommendation 10 February 2004, URL: http://www.w3.org/TR/rdf-primer/, last accessed: 14 July 2005.

[52] Ritter, J, (2001), Why Gnutella Can't Scale. No, Really, URL: http://www.mscenter.edu.cn/prj/files/127/Why%20Gnutella%20Can't%20Scale_%20No,%20Really.htm, last accessed: 28 February 2005.

[53] Robertson, S., (2004), Understanding Inverse Document Frequency: On

Theoretical Arguments for IDF, Microsoft Research, Cambridge, UK.

[54] Rouse, C., and Berman, S., (2005), A Scalable P2P Database System with Semi-automated Schema Matching, paper accepted to the International P2P/DAKS Workshop on P2P Data and Knowledge Sharing, to be held in Lisbon Portugal, July 2006.

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

[56] Shirts, M., and Pande, V., (2001), Screen Savers of the World Unite!, Science Magazine, Stanford University Libraries, Standford, USA.

[57] Stein, M., (2004), JXTA Technology: Creating Connected Communities, Sun Microsystems, Inc., Palo Alto, CA, USA.

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

[59] 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.

[60] Treloar, A., (2004), Design and Implementation of a Query-By-Example Interface for Micro CDS/ISIS, School of Computing and Mathematics, Rusden Campus, Deakin University, Clayton, Victoria, Australia.

[61] Unknown Author, (2003), Introduction to P2P networks, P2P networks project, http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/Intro.html, accessed on 20 April 2004.

[62] Wang, G., Goguen, J., Nam, YK., and Lin, K., (2004), Critical Points for Interactive Schema Matching, San Diego Supercomputer Centre, U.C. San Diego, San Diego, CA, USA.

[63] Waterhouse, S., Doolin, D., Kan, G., and Faybishenko, Y., (2002), Distributed Search in P2P Networks (IEEE Internet Computing, February 2002).

[64] Waterhouse, S., (2001), JXTA Search: Distributed Search for Distributed

Networks, Sun Microsystems, Inc., Palo Alto, CA, USA.

[65] WordNet: A lexical dictionary for the English language, URL: http://wordnet.princeton.edu, last accessed: 10 November 2005.

[66] Xu, L., and Embley, D., (2004), A Composite Approach to Automating Direct and Indirect Schema Mappings, Department of Computer Science, University of Arizona South, and Department of Computer Science, Brigham Young University.

[67] Yan, L., Miller, R., Haas, L., and Fagin, R., (2001) Data-Driven Understanding and Refinement of Schema Mappings, ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA.

[68] Yang, B., and Garcia-Molina, H., (2002), Comparing Hybrid Peer-to-Peer Systems, in Proceedings of the 27th VLDB Conference, Rome, Italy.

[69] Zaihrayeu, I., (2002), Simple JXTA, Sun Microsystems, Inc., Palo Alto, CA, USA.

[70] Zhou, A., Qian, W., Zhou, S., Ng, W., Ooi, B., and Tan, KL., (2002), Towards Data Management in Peer-to-Peer Systems: A Perspective of BestPeer, Department of Computer Science and Engineering, Fudan University, and Department of Computer Science, National University of Singapore.

Appendix

Mapster User Interface Evaluation Questionnaire

Information:
This is to test my system, Mapster, and particularly the interface. Please feel free to make any comments during the interview about either. The results are anonymous and will only be used to determine the effectiveness of the interface and nothing else. Thank you.

Mapster is a database-sharing P2P system. A P2P system is one that allows people to communicate with each other and to share resources, such as files, in an easy and relatively ad hoc manner. Mapster uses a P2P platform to share people's databases.

Mapster uses domains and schema matching. A domain is a group of peers that have similar interests. Their databases contain similar data. Schema matching is a technique used to determine how one database relates to another. It calculates mappings between databases so that queries can be translated from one database to the other.

Task 1:
Start Mapster and join an existing domain.

Ease (Please circle)
1 (easy) 2 3 4 (difficult)

Comments:

_____

_____

_____

_____

Task 2:
Browse the domain database.

Ease (Please circle)
1 (easy) 2 3 4 (difficult)

Comments / Problems:

_____

_____

_____

_____

Task 3:
Query the domain. Write a query in terms of your own database.

Ease (Please circle)

1 (easy) 2 3 4 (difficult)

Comments / Problems:

Task 4:
Query the domain schema directly.

Ease (Please circle)
1 (easy) 2 3 4 (difficult)

Comments / Problems:

Task 5:
Use the following query to obtain specific results from the domain.

SELECT age, ID FROM ms WHERE age > 18

Ease (Please circle)
1 (easy) 2 3 4 (difficult)

Comments / Problems:

Questions:
1) Overall, how easy did you find it to use the interface?

2) Was there anything that was unclear?

3) Was there anything particularly difficult to do?

4) Could you please describe a scenario where you think this system would be used?