Peer-to-Peer Data Integration Using Distributed Bridges

submitted by

Neal Arthorne, B.Eng.

A thesis submitted to
The Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario

April 26, 2005

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis:

Peer-to-Peer Data Integration Using Distributed Bridges

submitted by Neal Arthorne, B. Eng. (2002)
in partial fulfillment of the requirements for
the degree of Master's of Applied Science in Electrical Engineering

---

Chair, Rafik A. Goubran, Dept. of Systems & Computer Engineering

---

Supervisor, Babak Esfandiari

April 26, 2005
Carleton University

Carleton University
April 2005

Abstract

The traditional methods of data integration use a centrally administered global schema that is not well-suited to a large number of data sources that frequently change. Using the peer-to-peer approach, this thesis proposes the use of user-contributed mappings or 'bridges' that can be published and exchanged over a network of peers. The Universal Peer-to-Peer (U-P2P) framework is modified to allow bridges to be created and a schema definition for the bridges is presented that contains mappings and a semantic relationship. Large databases are connected to the peer network using proxy modules that allow U-P2P to connect to a live data source using a custom implementation. A fully decentralized network adapter is implemented in U-P2P to prevent large databases from having to upload their metadata to indexing servers. The proposed system is used to implement simple data integration between two digital repositories using query translation.

Table of Contents

List of Figures

List of Appendices

Chapter 1 Introduction

In today's highly connected world sources of data are used behind the scenes of many corporate information and knowledge management systems, web portals, e-commerce systems, search engines and countless other applications. To the end-user, the data is invisible and is only useful or noticeable when it is rendered onto a screen, typically in a web browser or custom application. Behind the user-interface a communication protocol is used to connect to a database, execute a search query, compile and then return the results. The web browser or application then displays the results and performs further less complex sorting and analysis. One model for the retrieval of data is the client-server model, in which an uncomplicated or 'thin' client such as a web browser connects directly to a database running on a corporate back-end server. The server takes care of crunching the data and answering queries for the user while the client provides the user interface and takes care of presentation. The client-server interactions are wrapped in layers of communication and security protocols to provide a relatively safe, secure and reliable connection to the data. Examples of such client-server systems are widespread and include most of the World-Wide-Web, point-of-sale and banking transaction systems, inventory control, and enterprise applications for resource planning.

Considering that end-users are usually unaware of the behind-the-scenes client-server mechanisms, what happens when they want to expand the scope of their latest data queries to include a new related data source? In the client-server model this cannot be achieved without intervention from the database administrator, the application developer

and other system administrators. For the client software, it can be as complicated as re-writing and re-deploying a custom application or as simple as pointing a web browser to a new address. On the server side, integrating a new data source may require extensive re-writing of code and numerous rounds of testing and re-design for the system to work. The introduction of new data may require new access controls and added layers of security that only serve to complicate access to the system and could cripple its usability and performance. For the most part it would be easier for the user to switch to an application that uses the new data source than to persuade the database administrators and software developers to change their application.

To further complicate accessing multiple sources of data, the data itself may be in a format that conflicts with existing data. Every database has a schema or model that is developed to suit its purpose and intended user-base. Conflicts may arise when an attempt is made to integrate two schemas and in some cases, a new schema must be written or a middle-layer of software added to mediate between the data sources and their respective schemas.

Instead of expanding client-server applications to include more data, we propose two major changes to the way users access their data. First, make every source of data a node or 'peer' on a network regardless of whether it is a client or a server and second, use mappings between the sources of data and make those mappings publicly available for use by anyone. By supporting a fully-distributed network of users who can all share and access data, the scale of the network and the number of data sources can far exceed the typical client-server model while maintaining a reasonable level of performance and usability. And by allowing users to create their own mappings or bridges between

different types of shared data, a fully-distributed web of relationships can be layered on top of the network and used when searching.

1.1 Motivation

The integration of autonomous, heterogeneous data sources can be a crucial advantage to both large and small-scale databases systems that operate in both private commercial systems and in public sector and academic research. There are many reasons to integrate data across multiple sources, the foremost being the desire of a user to search across multiple databases using a single query or query language and to return a result that contains data from multiple sources. From the user's perspective, it often seems like a natural thing to do, but for databases, integration of new data sources is labour-intensive and often requires surmounting large technical and political barriers. While the latter is outside the scope of this thesis, some of the technical challenges of integrating multiple data sources will be addressed.

By integrating databases, information can be extracted from multiple data sources in one fell swoop without learning how to query each individual database using different data formats, query languages and communication protocols. Alternatively, a user could manually query multiple databases and correlate the results by hand but for data sources that deal with large quantities of data, a manual approach of sifting through data is not feasible and negates the benefits of sharing data from multiple sources.

The demand for data integration has been recognized since the introduction of database systems and although the uses of integrated data may have changed over the years, it still remains a high priority for many organizations. The practices of data mining and data

warehousing are examples of recent uses of data that can require extensive data integration. Aside from integrating databases for the benefits of corporate knowledge and end-user queries, a larger challenge is to allow autonomous software agents to search across databases. With the advantages of speed and precision, software agents open the door to innumerable possible applications where agents can search, retrieve, filter and present information to a human user on a regular basis or for a selected query. To enable these possibilities, the work undertaken in this thesis lays the groundwork for software agents to transverse across data schemas and use high-level semantic constructs. The creation of these software agents and their integration into the peer-to-peer framework used in this thesis is left for future work.

1.2 Contributions

The primary contribution of this thesis is a system for relating and integrating data shared in a peer-to-peer network. It allows a user to have their search query translated and executed on new sources of data that use a different schema. Query translation and data transformations are achieved through user-contributed mappings between data schemas called 'bridges'. This thesis involved extending an existing peer-to-peer framework to allow for creating, searching and viewing of these bridges built between communities of related data. The resulting Universal Peer-to-Peer (U-P2P) [Muk02] [Art03] framework was also improved to serve as a proxy to live databases. Since many large databases are publicly accessible over the Internet, integrating them into U-P2P by way of a proxy significantly increases the amount of data available to user of the U-P2P system.

Bridges in U-P2P

User-contributed mappings or 'bridges' are proposed as a means to relate one type of data to another. Metadata in U-P2P is formatted as XML and an XML Schema [Fal04] is associated with a community of users. To relate data from one schema to another, a user creates a bridge that is itself an XML document that describes the relationship between the two schemas. The bridge is then published to the network in a 'Bridge Community' that exists for this purpose and other users can then search for and retrieve the bridge for use in their searches. Since a bridge also contains a semantic statement about its source and target communities, it can be used to create a web of semantic relationships over the existing data to be used when performing searches. Sharing bridges themselves in U-P2P further helps data integration by allowing any peer to publish a bridge between schemas instead of restricting the set of available bridges or using a centrally administered form of data integration.

U-P2P as a Database Proxy

The U-P2P framework was modified to allow a U-P2P peer to use a database that is external to the network. This allows a proxy to be set up for any existing database with data that can be translated into Extensible Markup Language (XML) [Bra04]. Since many major database vendors now provide tools to export data to XML or provide native access to XML data, the proxy has the potential to connect thousands of existing databases to a network of U-P2P peers. When combined with bridges, the proxy mechanism allows existing data to be related to data shared on the peer-to-peer network or available from other proxy nodes. A proxy module is implemented in U-P2P so that it may only service selected methods of the repository functionality. This allows a proxy to

operate in read-only mode if the data source is a read-only public database or the U-P2P user does not have access privileges for committing data. Should the proxy wish to commit data, it can work with the existing mechanisms in place for publishing resources to the peer-to-peer network.

If a proxy mechanism for live databases was not provided, integrating data in U-P2P would be restricted to data held in the local XML repository. Data could be copied or harvested from a live database into the local store for integrating in U-P2P, but this would require continuous updates as new data is entered and the freshness of data used in queries could not be guaranteed. By rewriting queries to the native database format, the proxy receives the latest data and avoids the problems of maintaining data freshness.

A Distributed Network Adapter

A fully distributed peer-to-peer protocol is essential for integrating numerous autonomous data sources in the U-P2P framework. It allows data sources to be truly independent of both servers and centralized indexes used in the initial peer-to-peer network adapter. The initial version of U-P2P provided an adapter for a generic version of the peer-to-peer protocol used in the now superseded Napster file-sharing service [Nap05]. The Napster-like protocol uses a centralized server that indexes XML metadata and responds to user search queries. While additional servers can be used, each server has a limited user capacity and the U-P2P adapter has no load-balancing features for deflecting incoming requests to underused servers. It was also not designed for users sharing large amounts of metadata that must be uploaded to the central server. To increase scalability and to limit the amount of indexed metadata exchanged between a peer and the centralized server, a new adapter was written using the fully-decentralized

Gnutella protocol [Gnu05]. Since existing large public data sources are often the target or motivation for data integration, developing a decentralized peer-to-peer protocol adapter was an essential part of extending the U-P2P framework for data integration. Without a decentralized adapter, large data sources would have to upload vast quantities of metadata to a centralized indexing server to participate as a node in a network of U-P2P peers.

It was also critical to develop the Gnutella adapter to allow proxies of live databases to connect to the network and not have to upload all the metadata for their database to the central server for indexing. Instead of uploading all its metadata, a peer acting as a proxy for a live database connects to the network and waits for other peers to send queries that it answers by searching the live database. If there was no decentralized peer-to-peer adapter, proxies of live databases would not be possible without first harvesting all metadata from a database and storing it in the local database operated by the peer. As mentioned in section 0, this presents problems with keeping the data up-to-date and can be avoided by using a live database proxy accessed by a decentralized peer-to-peer protocol.

Digital Repository Case Study

A case study using digital repositories is used to show the ease and flexibility of the upgraded framework in peer-to-peer data integration. Digital or institutional repositories are being deployed at universities, libraries and organizations around the world to serve as permanent storage for the growing body of electronic documents and media created by organizations. Many of the systems operate using repository software that uses XML-formatted metadata such as the Dublin Core metadata set [DCT05]. For example, a

faculty member at a university might upload an electronic document to a repository and provide an author, title, date of publication and subject keywords. The repository software takes these attributes and generates an XML record that is time-stamped, attached to the electronic document and stored and indexed in the archive. The two repositories used in the case study, Fedora and DSpace, both provide basic publishing and search services, but lack any integration features that make cross-repository searches possible.

With the improved U-P2P framework, the main features of these two repositories are implemented in a matter of hours without writing custom web applications or a federated database schema. After creating communities in U-P2P for each repository, the repositories are then integrated using a bridge that supports basic query translation and a semantic-level mapping between the schemas. Instead of taking hours to manually integrate the differing repository schemas and create a globally shared schema that is both fragile and difficult to administer, U-P2P demonstrates its advantages by allowing a user to create a bridge and publish it on the peer-to-peer network for immediate use in bridging the two repositories. The case study also shows the limitations of U-P2P in integrating data as the query translation uses a simple set of mappings that are not as powerful as the logic statements used in related work and some fragility remains in the mappings between data schemas.

The proposed solutions introduced in this chapter address some of the issues described in the next chapter on the challenges of data integration. While this thesis cannot address all of the issues in data integration, the proposed user-contributed mappings are a step toward a highly connected and related web of information shared between peers on a

network. The peer-to-peer approach combined with access to existing large databases and a fully-distributed network topology provide a powerful tool for integrating data on a large scale. With the addition of semantic-level relationships contained in the mappings, the U-P2P framework will also serve as the basis for future work on semantic data integration.

1.3 Thesis Structure

After introducing the motivations and contributions in this chapter, Chapter 2 will provide an overview of the current research related to data integration and specifically to peer-to-peer approaches to data integration. Chapter 3 will outline the software framework used in the proposed solution and Chapter 4 offers advantages and alternatives that should be considered when basing the solution on U-P2P. The data integration solution including the definition of a bridge and its community is explained in Chapter 5 and Chapter 6 contains a case study to demonstrate the use of the bridges in a concrete scenario.

Chapter 2 State of the Art

The integration of autonomous, heterogeneous sources of data is a widely researched field that has changed over the course of the years to reflect new data formats and new methods of providing access to data. The following sections outline the challenges of data integration and the generations of solutions that have been developed to address them. This thesis offers solutions that relate to these techniques and the related work on peer-to-peer data integration presented later in this chapter.

2.1 Challenges of Data Integration

The main challenges are threefold and deal with the differences in distribution, autonomy and heterogeneity [She99]. The following sections will discuss the challenges of integrating data, the different generations of techniques that have been developed to meet those challenges and the work related to peer-to-peer data integration.

2.1.1 Distribution

Distribution is primarily concerned with communication between multiple physically separated data sources. Thus the databases may need to be accessed by specific protocols or networks across a geographically wide area with implications for security, bandwidth requirements and network latency. The use of the Internet and the TCP/IP protocol stack has largely reduced this problem in terms of accessing databases in physically remote locations, but the challenges of security, network latency and bandwidth requirements remain. Database security and networking issues are outside of the scope of this thesis,

but this work will address network topologies for data access and how to integrate data in a distributed environment of networked data sources.

2.1.2 Autonomy

Autonomy is another primary source of differences in databases. Organizations that create and maintain data sources prefer to remain in control of all aspects of their databases. Any integration project must start by addressing who will gain or lose control of the data and if any, what aspects of autonomy need to change [She99]. In the context of this thesis, databases remain autonomous or independent of each other and may be integrated through their own participation in a data integration scheme or by a third party acting as a proxy to the database without their knowledge. In either case, it is assumed that the database access controls can be programmed to allow access either by the general public or specific users that will integrate the data.

Autonomy of database design is one of the biggest challenges of data integration because it implies that each database is designed differently and with different needs, requirements and end users in mind. The differences in design philosophy often produce the low-level differences in data formats, databases schemas, access control and other functionality [She99]. For example, a biological database may be designed for a small department of a private research company while another may be used by medical staff at a hospital. Although they may contain similar types of information, the data sources are likely to be quite different due to the autonomy and independence of the owners.

The original database may also be heavily used by its current set of users and a data integration project may require additional capacity or computing power to make sure the

original users can continue to use their database. In some cases, the database operator might move their users to the integrated system and completely change how their data is accessed.

Autonomy in the design and use of databases gives rise to many challenges as described above, but this thesis will only address the technical aspects of integrating the data and not the issues with user capacity, migration, access control or the security of individual databases.

2.1.3 Heterogeneity

Autonomy spawns the heterogeneity found in multiple, related data sources. Sheth identifies four kinds of heterogeneity [She99]:

- System – differences in Database Management Systems (DBMSs), operating systems, hardware requirements and system capabilities
- Syntax – data may be represented in a machine-readable format that is different in each database
- Structure – differences in database schemas and data models (e.g. relational versus an object-oriented)
- Semantic – differences in the meaning or intended use of data fields stored in each database (e.g. a letter grade for a course in one school is different from a percentage grade for a course in another school, but they have a semantic relationship)

The first two types of heterogeneity have been largely addressed by previous work in the field of data integration and the last two challenges are the focus of this thesis. Syntactic data differences for example, are largely solved by using an interchangeable and widely accepted data format such as Extensible Markup Language (XML) [Bra04].

2.2 First Generation Approaches

In the 1970's databases were incompatible and difficult to integrate because of differences in operating systems, hardware and communication links [She99]. Although many of these problems were overcome with standardized protocols, software and communication, some challenges persist and continue to plague the current generation of data integration techniques.

The 1980s introduced a pressing need to integrate structured data and Database Management Systems. Sheth and Larson in [She90] and Litwin in [Lit85] and [Lit90] pioneered federated and multi-databases that dealt with system interoperability and differences in data structure, data constraints, query languages and transaction management. Researchers at the time focused on data as opposed to knowledge or information and despite the progress made, few commercialized products resulted from what Sheth calls the first generation of data integration [She99]. One of the major contributions was federated database systems and the approach proposed in this paper can be considered as a federated database system, albeit on a much larger scale and with no globally shared schema.

2.2.1 Federated Database Systems

A federated database system (FDBS) presents one or more uniform interfaces to a set of structured component databases or DBMSs [She90]. This approach was developed as a methodology that focused on integrating relational databases using one or more shared database schemas. Two types of FDBSs are identified by Sheth and Larson [She90]: tightly and loosely coupled systems.

A tightly coupled FDBS allows one or more federated schemas to be used to access all shared component databases. The federated schema is controlled by an administrative body that works with component databases to ensure interoperability at the semantic and structural levels. Tight coupling is used when the component database schemas are relatively static and do not change. It is also used when data needs to be closed guarded or is an important resource that should only be queried or receive updates through one federated schema. A tightly coupled FDBS can also use multiple federated schemas but this increases the difficulties in maintaining interoperability as each component database must maintain mappings to multiple federated schemas.

Loosely coupled FDBSs offer no single federated schema but instead allow the users to create their own schema to access the data. Because there is no federated schema, users are required to have a much greater understanding of database components, query languages and database-related skills.

2.3 Second Generation Approaches

The second generation of data integration is characterized by support for a larger variety of data sources, use of metadata to support integration and use of knowledge

representation to represent data [She99]. One of the techniques from this generation is the mediator approach.

2.3.1 Mediator Approach

The mediator approach provides a layer of interoperability between a global, mediated schema seen by the user and the local schemas used by the autonomous component databases. An example implementation is TSIMMIS [Garcia-Molina95]. It wraps each database in a layer that translates queries and results to and from a common information model. The translation modules talk to a mediator module that receives user queries for a specific domain and combines query results to present to the user. The mediator modules are created using a high-level description and can communicate with other mediators because they present one interface to translators, users and other mediators. Mediators usually work at a higher level of abstraction that deals with information and knowledge gathering, rather than just data [She99].

2.4 Third Generation Approaches

The proposed data integration scheme falls into the third generation of approaches to data integration which focuses on integrating semantic differences as well as differences in data syntax and structure. The emphasis is on knowledge and information and with it, the possibilities of inference and reasoning. Reasoning can be used to generate new information that is not directly contained in a set of documents by exploiting the known properties of relations within the data. If data can be understood at a higher level, it becomes much more useful and can be related to other data at a semantic level to allow high level queries. A high level query resembles natural language more than query

languages such as SQL. However, a shared vocabulary for our data is needed when attempting to answer high-level queries.

2.4.1 Ontologies

An ontology is a vocabulary for describing a specific domain of information [Hef04]. It is the concepts and relations between concepts that need to be understood to function in a domain. For example, an ontology in the domain of academics might describe the concepts of a course, a student, a professor, a test or assignment and a physical location where learning takes place. Natural language can be used to describe these concepts, but for the purpose of allowing computers to understand ontologies, they must be described using a machine-readable language. Ontologies will be used in data integration to provide a model at a semantic level of the shared data. With an ontology or vocabulary that describes concepts in a set of data, we can create mappings into other ontologies and describe the relationships embodied in the mappings using predefined terms. For example, an equivalent mapping can be used to map the term 'Author' in one ontology to the term 'Creator' in another ontology. Once this mapping is established, it is possible to use automated tools that explore the mappings and allow a query to be translated and answered by data that uses a different ontology.

One of the drawbacks of using ontologies for data integration is that the mappings between two ontologies may be incomplete or incongruent. The problem of re-alignment of ontologies is discussed in 2.4.5 after introducing a standardized ontology language based on the Resource Description Framework (RDF).

2.4.2 Resource Description Framework

A first step towards describing a vocabulary or an ontology for a domain is the use of a language to describe the relationships between concepts or resources. Such as language, as promoted by the World-Wide-Web Consortium (W3C), is the Resource Description Framework (RDF) [Kly04] [Man04]. RDF describes relationships between resources on the World-Wide-Web. It can be visualized as a directed graph using a *subject*, a *predicate* and an *object*. For example, in Figure 1 a subject denoted by the Uniform Resource Identifier (URI) *http://www.example.com/mypage* can have a predicate '*Author*' and an object '*John Doe*'.

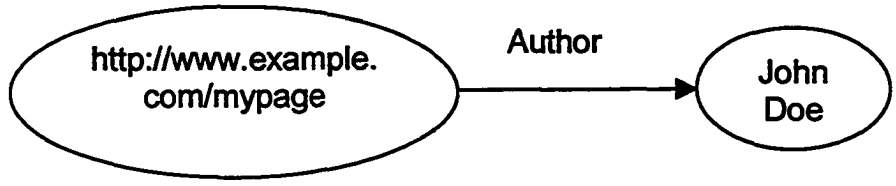

Figure 1 Example RDF directed graph.

Using the above model, an assertion can be made about a resource that can be uniquely identified with a URI. A collection of RDF statements can be used to model information or processes and then used by a reasoning engine to entail new information from the RDF. The process of entailment or reasoning extracts new information from a collection of logical statements that is not explicitly stated. RDF is also geared toward interoperability to allow applications to process RDF from many different sources and derive new information [Kly04]. As such, RDF has an XML syntax that can be processed by machines using standard software tools.

It should be noted that while RDF can be used to describe relationships between web-based resources, the URI that identifies an object is only for identification and does not imply that anything resides at this address on the network. This method of using URIs is like that of the Namespace mechanism of XML [Bra99], where a URI is associated with a namespace prefix. URIs in RDF are used to describe any manner of objects, some of which are physical objects that can only be symbolically represented in a graph (e.g. employees, addresses).

2.4.3 RDF Schema

RDF describes relationships between objects but has no provisions for creating a vocabulary of objects that can be consistently re-used when composing new graphs. This is where RDF Schema [Bri04] steps in and allows the creation of full vocabularies. In RDF Schema (RDFS), classes and instances are defined that are analogous to classes and objects in object-oriented programming.

2.4.3.1 RDFS subClassOf and subPropertyOf

RDFS defines the properties `rdf:subClassOf` and `rdf:subPropertyOf` to designate that an instance of a class or property is also an instance of another class or property. Thus RDFS introduces inheritance to RDF classes and properties and the associated benefits of generalization.

2.4.3.2 RDFS domain and range

The example in Figure 1 of an author and a website, is created with a Document class and an 'Author' Property. The type and application of the 'Author' Property is given as its 'domain' and 'range'.

The domain of a Property specifies the classes to which the property applies. Thus if a Property 'Author' has the domain 'Document' then any class that uses the Property 'Author' must be of class 'Document'. This differs from the object-oriented approach where a class defines a field or member variable, but the field itself does not imply anything about the class.

The range of an RDF Property is the class of objects that can be used as the value of the property. In our example, the range of the 'Author' property will be 'string', to indicate that the value of the property should be a string of characters. This is equivalent to the data type of a field in object-oriented programming.

Using RDF/RDFS in their XML syntax, the RDF Schema for the relation in Figure 1 is as follows:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.com/">

<rdfs:Class rdf:ID="Document"/>

<rdfs:Datatype rdf:about="&xsd;string"/>

<rdf:Property rdf:ID="Author">
  <rdfs:domain rdf:resource="#Document"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

</rdf:RDF>
```

Note that the 'Author' property in the schema is defined independently of the class definition of a 'Document'. This allows reuse of the property by other classes within the schema or by external documents. It is also conducive to a distributed approach to

building RDF graphs because anyone can add a property to a class without changing the original definition of the class.

The following is an instance of RDF that uses the RDF Schema given above and represents the relation shown in Figure 1:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.com/schemas/webSchema#"
            xml:base="http://example.com/">

  <ex:Document rdf:ID="mypage">
       <ex:Author rdf:datatype="&xsd;string">John Doe</ex:author>
  </ex:Document>
</rdf:RDF>
```

Note that the RDF Schema is referenced using the URI for the namespace 'ex' and can similarly be used in further RDF statements or in other RDF documents. This allows the linking of one RDF Schema to another, which is significant for our use of the OWL language described below and based on RDF/RDFS.

2.4.4 Web Ontology Language

The Web Ontology Language (OWL) [McGuiness04] [Pat04] builds on top of the Resource Description Framework and RDF Schema. OWL augments RDF to describe relationships between objects such as equivalence and cardinality. OWL, along with RDF and RDFS, can be expressed in an XML syntax [Bec04].

The position of OWL in the language 'stack' or layering used by the World-Wide-Web Consortium for the development of the Semantic Web is shown in Figure 2 and is described in [McG04]. The Semantic Web is a movement towards a machine-readable

web that separates content from presentation and allows software agents to reason and extract knowledge from the web [Ber01].

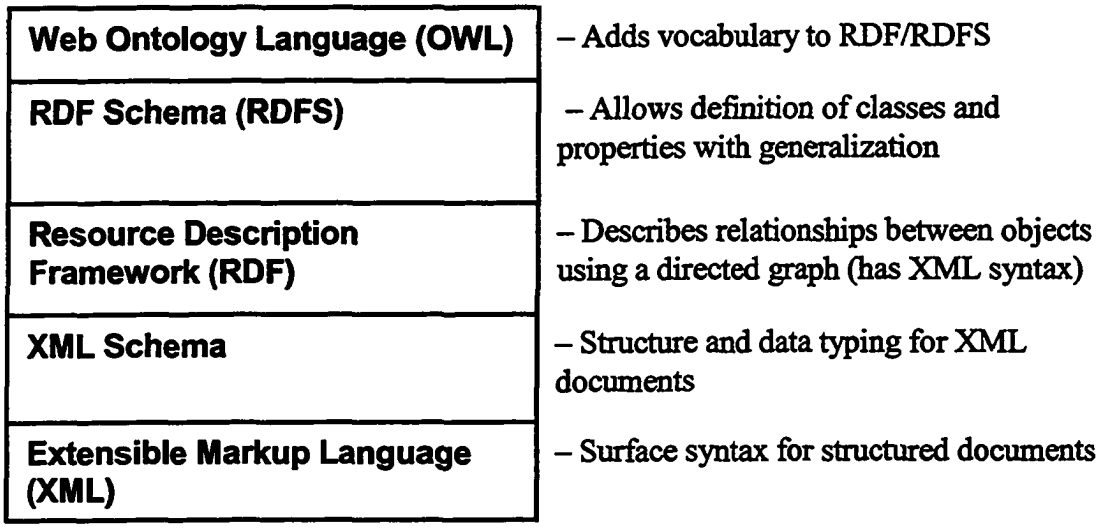

Figure 2 W3C Semantic Web language stack

2.4.4.1 OWL Additions to RDF/RDFS

Similar to RDFS, the concept of a class is central to OWL. Instead of instances, OWL uses the term 'Individual' to describe an object that is a member of a class. A class in OWL is a collection of properties that describe a set of individuals, and an individual is a member of that set [McG04]. A class contains a name and a set of restrictions that relate the class to other classes (as in RDF Properties).

OWL augments RDF/RDFS by adding constraints for cardinality of class relationships (e.g. a *Person* may have exactly one property of class *Mother*), and property relationships (e.g. a *Document* has one or more *authors*). It also adds constructs for specifying equivalence between classes (`owl:equivalentClass`) and equivalence between individuals of a class (`owl:sameAs`).

OWL also adds explicit statements for making a Property transitive, symmetric, functional or inverse functional. Thus when a Property is known to be symmetric, for example where A has B implies B has A, it can greatly improve the reasoning over the set of OWL/RDF statements that use the symmetric Property. New relations can be discovered just by knowing that a single Property is symmetric and OWL provides the means to explicitly support those efforts.

2.4.5 Ontology Alignment

Given a powerful language like OWL, ontologies can be described for a domain and relations between objects can be identified and used for reasoning, but a problem arises when two ontologies for a semantically-related field are compared and they do not match. This is a similar problem to differences in schemas that exist at the structural level of heterogeneous databases. One database may have a specific definition of a concept while another may differ in opinion or may try to simplify a definition to suit their needs. When two schemas or ontologies do not match they are said to be out of alignment.

Aligning ontologies has received recent attention such as [Doa03] [Pin01] and [Noy00] and it is an important step towards semantically-enabled database and information systems and for realizing the vision of the Semantic Web. If two ontologies share similar concepts but are not aligned, it is very difficult to take advantage of semantic reasoning or inference and to create queries that span both ontologies. If the two ontologies belong to domains that are orthogonal or independent, then aligning the ontologies may not be feasible and further higher-level alignments may first be necessary. Assuming the domains are not disparate, there are several different methods for aligning ontologies.

A common approach is to manually align the ontologies within a domain using the help of domain experts. Cruz et al. used this approach in the Wisconsin Land Information System [Cru02] that aligned definitions of land use codes that varied from county to county. Experts from each county went through the land use codes and integrated them with a state-wide model. The mappings from county codes to state-wide codes were then published in a 'declarative agreement' using a proprietary format.

Cruz also proposes a semi-automatic alignment method in [Cru03]. Given an ontology tree structure, one non-leaf node, preferably close to the root, is manually aligned to the second ontology. From here, an algorithm attempts to align the two ontologies based on a limited set of relations used in the tree such as subset, superset, exact and approximate matches. This process requires manual intervention and correction for nodes that cannot be matched and only uses a fixed set of relations that does not approach the expressive power of a language like OWL. Cruz et al. also use a simplified ontology that only contains a single class per node without properties or other restrictions.

2.4.6 Logical Descriptions of Data Integration

In [Len02], the data integration problem is described from a theoretical viewpoint using logical descriptions of data sources and their relationship with a global schema. A data source is a set of relations (i.e. rows in a relational database table) and each component database has its own local schema describing its data. The global schema may be federated or mediated as described in 2.2.1 and 2.3.1, and provides an integrated view over the data. Mappings from local schemas to the global schema (often written as Datalog statements) are established in one of two ways: global as view (GAV) or local as view (LAV).

2.4.6.1 Global as View

In a GAV approach the elements of the global schema are defined as views over the local schemas. A view is simply a named query or 'virtual' relation that is created from existing database relations. For example, a GAV mapping might be the union of a relation from one schema with the relation from another. By defining the global schema in terms of views, answering queries is very efficient and can use simple unfolding techniques that substitute relations in the global schema with their corresponding views in the local schemas. A disadvantage of GAV is that the local schemas must be stable and should not change frequently. If they do change, elements of the global schema will be affected and views may need to be redefined to re-establish a complete description of the global schema. Examples of GAV style mappings include those used in TSIMMIS [Gar97], Garlic [Rot96], COIN [Goh99] and MOMIS [Ben00].

2.4.6.2 Local as View

The local as view approach does the opposite of GAV and defines the local schemas as views over the global schema. Thus in LAV, each element of the local schema is defined as a query over the global schema. This leads to problems when the global schema does not have enough information to describe the relations in the local schema and the result is that local schemas may be incompletely described. When answering queries with LAV mappings, techniques of view-based query answering are employed. View-based query answering is useful in other areas of database systems and much research devoted to the topic including [Hal01], [Abi98] and [Dus97]. An advantage of LAV mappings is that the source databases can change their schemas without disturbing the global schema. The global schema, on the other hand must remain relatively stable and unchanged. If the

global schema were to change, each database might have to redefine its views over the global schema. Thus LAV is suitable when there are many data sources that frequently change and one global schema that remains stable. Such would be the case when an enterprise uses a globally shared ontology to which all local databases relate.

Both LAV and GAV mappings are used in a variety of data integration projects and logical descriptions of data integration are often used when attempting to prove the complexity of answering a query across the global schema. GAV and LAV mappings may be combined as GLAV as seen in [Fri99], where both types of mappings are used to form a more expressive mapping language.

2.4.7 Peer-to-Peer Data Integration

A peer-to-peer networking approach to data integration tackles the problems of distribution and autonomy by making each node on the network equal and independent of its peers. Servers become clients and vice versa, with the effect that no single node controls the network and the addition or removal of a node will not adversely affect availability of data.

Aside from distributing data across a set of peers, the peer-to-peer approach has several implications with respect to data integration. First, a globally shared schema is eschewed in favour of each peer maintaining a schema for its own data. Thus, there is no global schema that must be centrally maintained as in federated databases, or extra layers of translation software as in mediator-based systems. Instead, mappings are from the local schema or peer schema to schemas used by other peers. This is a great advantage in large data integration systems that use many different peers in a dynamic network

environment. With no central schema to modify, peers can join the network, advertise their schemas and establish mappings to new schemas. A disadvantage is that mappings between peer schemas may quickly become outdated or invalid when a peer disconnects from the network or modified their schema. Also, the mappings have to be exchanged over the network or stored in a central location where they can be used in query translation. Each peer will use the mappings to reformulate their queries before sending them to the network, or alternatively peers could accept queries over different schemas and perform the query translation on behalf of the peer that sent the query.

2.4.7.1 Logical Peer-to-Peer Data Integration

A further challenge to data integration in peer-to-peer systems is providing any assurance that a query in a given schema will return complete or accurate results in a finite period of time. To formalize the problem, Lenzerini presents a framework in [Lenzerini04] for peer-to-peer data integration using logical descriptions that are like those in section 2.4.6. The framework describes each peer as holding a set of relations (given in Datalog or first order logic (FOL)) that describe the data they make available on the network. Each peer has a local or peer schema and holds mappings to other peer schemas to be used in query translation. This is the same as formal logical description of data integration, but lacks the globally shared schema.

Having described peer-to-peer data integration in FOL, Halevy et al. use it in their proposed Piazza Peer Data Management System (PDMS) [Hal03a]. In [Hal03b], the authors describe their mapping language used in mediating or integrating peer schemas and demonstrate the decidability of queries in their system. FOL mapping languages are also used in [Ber02], [Fra03] and [Cal04], with the later proposing the use of epistemic

logic. One of the drawbacks of FOL mappings is that it does not support queries over cyclical mappings. A cyclical mapping is where a relation in one peer schema is also in another peer schema creating a loop that does not allow a query to be resolved. To allow cyclic mappings Franconi et al. use a different logical model in [Fra04] and in [Cal04] the authors reformulate the model using epistemic logic for the same purpose.

In several of the above systems, complete knowledge of the network is assumed and all mappings are centrally stored for use in computations. This limits the use of these systems in a practical environment where no centralized store is used and networks are dynamic with nodes connecting and disconnecting continuously. Centrally locating the schema mappings also has the problem of federated databases where mappings quickly become outdated when component database schemas change. Current research is meeting these challenges with new distributed algorithms both for query answering [Fra04] and for storing and retrieving peer schema mappings using techniques drawn from peer-to-peer file-sharing.

2.4.7.2 Integration in File Sharing

The recent surge of interest in peer-to-peer systems has come about through the popularity of file-sharing applications such as the first generation of Napster, LimeWire [Lim05] and Kazaa [Kaz05]. Peer-to-peer file-sharing allows end-users to upload and download files from other semi-anonymous users connected to the same peer-to-peer network. Files shared by a user are usually presented in a flat list with no structure or relationship between files. Searches across the network may target specific metadata fields such as artist name and song title for music files, but there are no standards for metadata used by all systems and queries often fall back to file name searches. In some

systems, such as LimeWire, metadata about requested files is added to a query as an optional message block to be parsed and used by compatible clients [Tha01]. This allows for richer searches of typed metadata (i.e. find audio with bitrate greater than 192 kbps) but it does not integrate searching across multiple file types or searching across a specific domain. Although peer-to-peer file-sharing does not approach the complexity of peer-to-peer data integration, some of its techniques for fast retrieval of the location of files (such as distributed hash-tables) are being used in peer-to-peer integration systems.

2.5 Conclusion

The large number of approaches to data integration clearly indicates that there are many different ways to integrate different types of data and with various degrees of success. As database systems have evolved and changed, so too have the solutions for integrating their contents. From federated schemas and mediators to high-level ontology integration, the current body of work is highly focused on particular needs of a given information system or data integration project. Most projects start with a handful of databases whose schemas are known and whose access rights have been secured. They are typically not designed to operate in a dynamic network environment where data sources are frequently changing. To cope with the increasing number of data sources and their changing configurations, peer-to-peer data integration has seen new research interest. Formal logical models have been proposed and complex algorithms and mapping languages are used to describe relationships between data. The peer-to-peer approach eliminates the bottleneck of the centralized global schema and scales up to a larger, fully-distributed and networked environment of diverse data sources. It also brings web-like expansion to data integration by allowing any peer to publish mappings between peer schemas.

Although peer-to-peer techniques have been introduced to avoid the need for a centrally administered and potentially fragile global schema, they have their own drawbacks that must be taken into consideration. Namely, peer-to-peer data integration systems lack a global view of the data they seek to integrate and can only integrate two sources of data at a time, mapping from one peer to the next. While this eliminates the global schema, it does not make the individual mappings from one schema to another any less fragile than a local-to-global mappings used in a federated system. However, peer-to-peer has an advantage in that if a mapping between schemas is broken, a new one can be written and published on the network in much less time than it would take to modify a federated schema.

With logical models for data integration LAV, GAV or GLAV style mappings go a long way toward integrating data and provide a powerful and expressive mapping language. However, these mappings are limited to conjunctive queries or views that cannot express all the semantics of high-level concepts expressed in the data. A language like OWL is needed to describe ontologies or the vocabulary within a domain and to relate terms in a schema so they can be exploited with more complex reasoning and inference.

This thesis describes work done to extend the Universal Peer-to-Peer (U-P2P) framework [Muk02] [Art03] to allow integration of data using both schema-level and semantic mappings between communities of shared data. It improves upon the logical mappings of peer-to-peer data integration by providing high-level OWL mappings between the communities of data. It also allows any third party to contribute a schema mapping through the peer-to-peer network. This brings the expandability of the World-Wide-Web

to data integration and provides access to up to date mappings that can be applied without the need to modify any existing peer.

The following table provides an overview of the different data integration methods available and how the work completed for this thesis is related to other work in the field:

	Schema Location	Mapping Type	Mapping Format	Location of Mappings	Semantic Integration
Federated databases	Centralized	GAV, LAV	Proprietary	Centralized	Manual
Mediator	Centralized	GAV, LAV	Proprietary	Centralized	Manual
P2P Data Integration (e.g. PIAZZA)	Distributed	one schema to one schema	First Order Logic	Centralized	Manual
U-P2P Distributed Bridges	Distributed	one schema to one schema	XPath mappings (as yet)	Distributed	RDF/OWL statements

Each of the above approaches has advantages (such as a global view of data in federated or mediated approach) and disadvantages (such as a lack of semantic integration) that influence the method chosen to integrate data. Having described the above approaches the following chapter will outline the U-P2P framework which is relevant to the proposed approach that uses bridges distributed with U-P2P.

Chapter 3 U-P2P Background

Prior work in the field of peer-to-peer data sharing has resulted in the Universal Peer-to-Peer (U-P2P) framework [Muk02] [Art03]. The U-P2P framework provides an existing infrastructure suitable for building a peer-to-peer data integration system. The following sections will first give an example of the methods currently used to share data in peer-to-peer systems and then introduce U-P2P.

3.1 Sharing Data in a Peer-to-Peer Network

We propose using a peer-to-peer model for integrating data and this section describes some basic network topologies used in peer-to-peer applications. The primary advantage of peer-to-peer is redundancy of data. Redundancy increases availability and the speed at which data or resources can be retrieved from the network. Instead of having a central server that gets flooded with requests from clients, data transfers take place between individual peers and the availability of a file will increase as more peers retrieve a copy of the data and share it on their system. Since every node is a client and server, the joining or leaving of a node will not adversely affect the availability of a file, provided that there are enough copies stored throughout the network.

The advantages of a peer-to-peer network depend largely on the topology used by the peers. For example, a central server model (see Figure 3) was once used by the popular Napster file-sharing service [Nap05]. It employs servers that index all the files shared by peers and answer queries using keyword matching on file names. When connecting to the network a peer would send a list of all its shared files to the central server. The central

server would enter the list of files in a database and answer queries from peers on the network. A query would contain keywords that were matched to file names and a query response would direct the peer to download the matching files from other peers on the network. Since the central servers were owned and operated by Napster, the scale of the network could only grow in proportion to Napster's ability to provide the resources of bandwidth and computing power needed for the servers. The network was also easy to interrupt by removing the central servers and leaving the peers with no means to find each other (i.e. peer discovery) or route queries.

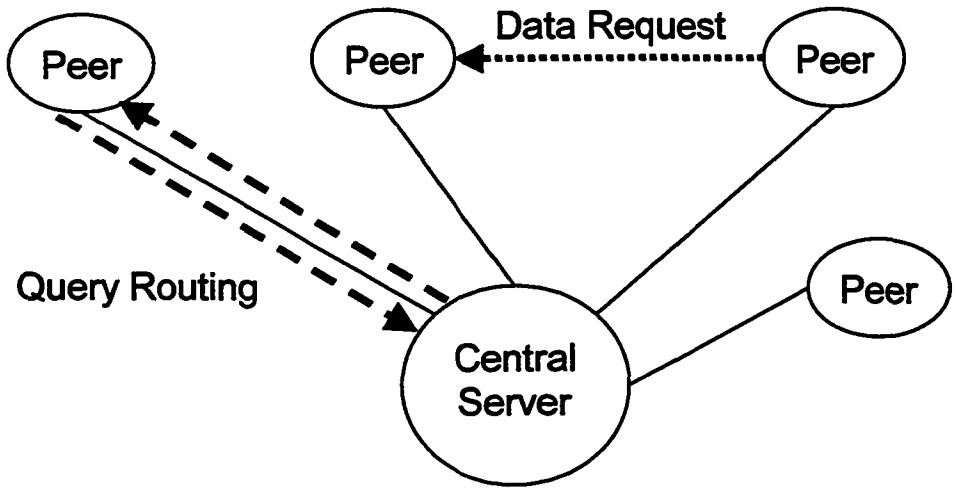

Figure 3 Centralized server model for a peer-to-peer network

To counter the disadvantages of a centralized indexing server, further peer-to-peer protocols were developed that provided fully distributed peer discovery and query routing. One such protocol is Gnutella [Gnu05] and it uses queries flooded across the network (see Figure 4). A peer in Gnutella maintains a small number of connections to its neighbours and when it originates a query, it tags it with a limit on the number of hops that it should make before terminating. This allows a query to spread across the network

to a limited number of peers, all of whom send any relevant replies to the originating peer and decrease the hop count of the query before passing it on. Routing query responses is handled by lookup tables that track where query messages came from and if a query has already been answered. A drawback of the Gnutella protocol is that discovery messages must be sent to maintain connections to neighbours and this may use up bandwidth that could otherwise be used for searching and retrieving files. Gnutella searches do not span the entire network (as this would flood the entire network with traffic) and responses to queries are usually slow and are not guaranteed to find a file.

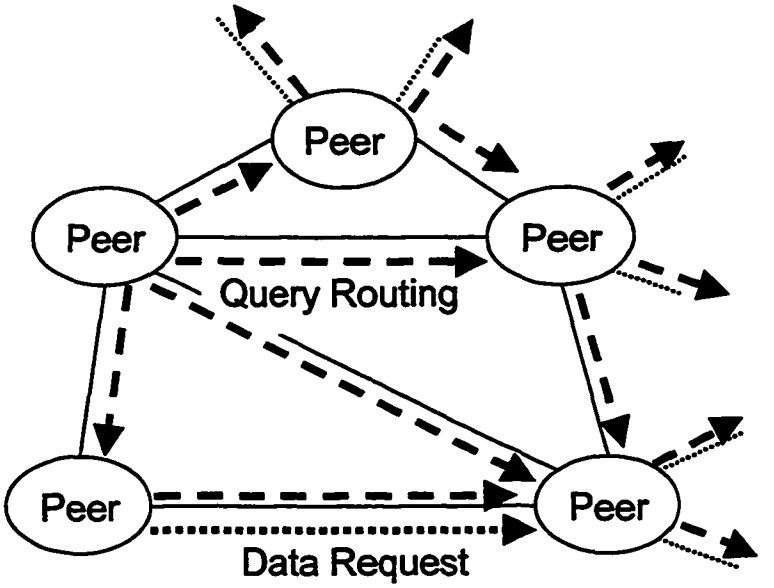

Figure 4 Distributed topology for a peer-to-peer network

Regardless of the network topology and its related protocols, peer-to-peer systems can distribute data across a network and make it reasonably accessible through a search protocol. Peer-to-peer file-sharing protocols such as Gnutella and Napster were created for use on public networks with thousands of clients and shared files and are not currently

used for integrating sources of heterogeneous data. However, their topologies are relevant to data integration and the lessons learned from large public peer-to-peer networks can be applied to integrating smaller numbers of large data sources.

3.2 Universal Peer-to-Peer (U-P2P)

U-P2P was developed by Babak Esfandiari, Aloke Mukherjee and the author, in the Department of Systems and Computer Engineering at Carleton University. It provides a framework for sharing data described in Extensible Markup Language (XML) [Bra04] and shared over any supported peer-to-peer network.

The core concepts of U-P2P are essential to understanding the contributions of this thesis in extending the framework for peer-to-peer data integration. A full treatment of U-P2P and its inner workings can be found on the U-P2P website (http://u-p2p.sourceforge.net).

At its core, the three shared components in U-P2P are *resources*, *communities* and *attachments*.

3.2.1 A U-P2P Resource

A resource is a data record that is written in XML, conforms to a specific XML Schema, and belongs to a community. It may contain metadata that describes a physical object, such as a book, or a network-accessible object such as a website. For example, in a digital repository a record could describe a scientific paper that is attached to the record in electronic form.

A resource may also be a data record itself and may not reference an attachment or external entity. For example, a resource may be a name and address that conforms to a schema for address book entries.

3.2.2 A U-P2P Community

A community is a combination of an XML Schema to which all resources shared in the community will conform, stylesheets to display the shared data, keywords to describe the community and a Network Adapter that determines the type of peer-to-peer deployment used by the community. The Network Adapter included with U-P2P uses the Napster style of peer-to-peer networking. A community itself is a resource and is published in the Root Community. This allows for the discovery of communities in the same way as any other resource.

3.2.3 A U-P2P Attachment

An attachment is a file that is bundled or associated with a resource. If a resource is downloaded from a peer, so are its associated attachments. Thus, as mentioned above, a resource can serve to describe an attached object or it can have no attachments and simply contain a data record. Because attachments are binary objects associated with a resource, they cannot be searched like resource metadata. Instead, users search through the metadata in a resource that describes the binary attachments.

3.2.4 The Role of U-P2P Communities

Communities in U-P2P can be thought of as a group of nodes that share the same type of object. The community founder writes a description of their community that includes the XML Schema they will use for the shared resources. Thus a community corresponds to

one XML Schema. If everyone wants to share the same objects, they download the community description and its schema and start sharing. Figure 5 shows an example resource and its related parts.

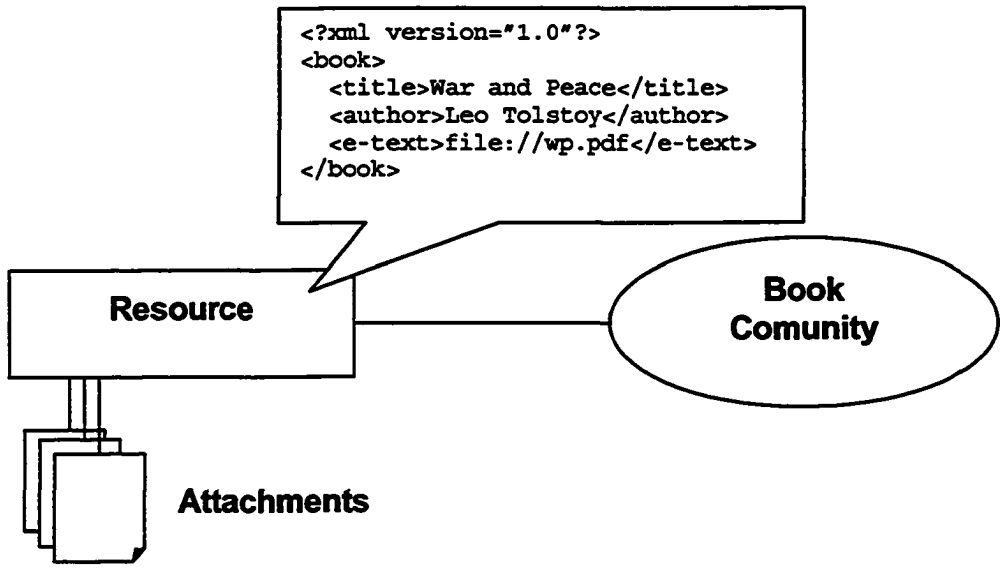

Figure 5 An example of a U-P2P resource.

The relationship between a resource and community is similar to the concept of objects and classes in object-oriented programming. A community is a class and it instantiates resources that are objects. Users can create, search and display resources (objects) as long as they have the template for the community (i.e. the class of the object is known).

Creating new communities gives rise to the question of how to discover new communities. U-P2P solves this problem by using a bootstrap Root Community that is provided so users can search for and join other communities. The Root Community can be centrally located or fully distributed, depending on the initial network configuration of U-P2P.

The advantage of sharing communities just like any other resource is that anyone can create their own peer-to-peer network. For example, if a user logs on and sees no suitable community for sharing their data, they can create a schema, write up the description and start a new community. As long as they publish the community to the Root Community, other users will be able to join it and start sharing data.

3.2.5 Network Protocols

The underlying peer-to-peer network layer is handled in U-P2P by plug-ins called Network Adapters. A Network Adapter is responsible for providing a means to search the network, retrieve a file from the network and publish a file to the network. The way in which these operations are completed depends on the implementation of the adapter. For example, if a user is searching a centralized network (as in the Napster style), the adapter would send a search query to the central indexing server and receive replies back through the adapter.

Each community has one Network Adapter and U-P2P provides a generic peer-to-peer adapter based on the centralized Napster-style model described in section 3.1.

In similar fashion to communities, Network Adapters are described in XML and shared on the network like any other resource. This adds flexibility to U-P2P so that it can download and install Network Adapters at runtime.

3.2.6 U-P2P Implementation

U-P2P is implemented as a web application that runs on the computer of a U-P2P user. It is written in Java and makes extensive use of JavaServer Pages, Java Servlets, and XML tools for presenting the user interface. The three primary facilities provided by U-P2P are

create, *search* and *view*. Each community supplies display stylesheets or HTML web pages for the create, search and view pages or uses the default stylesheets included with U-P2P. Figure 6 shows the JSPs and Java Servlets used to implement U-P2P.

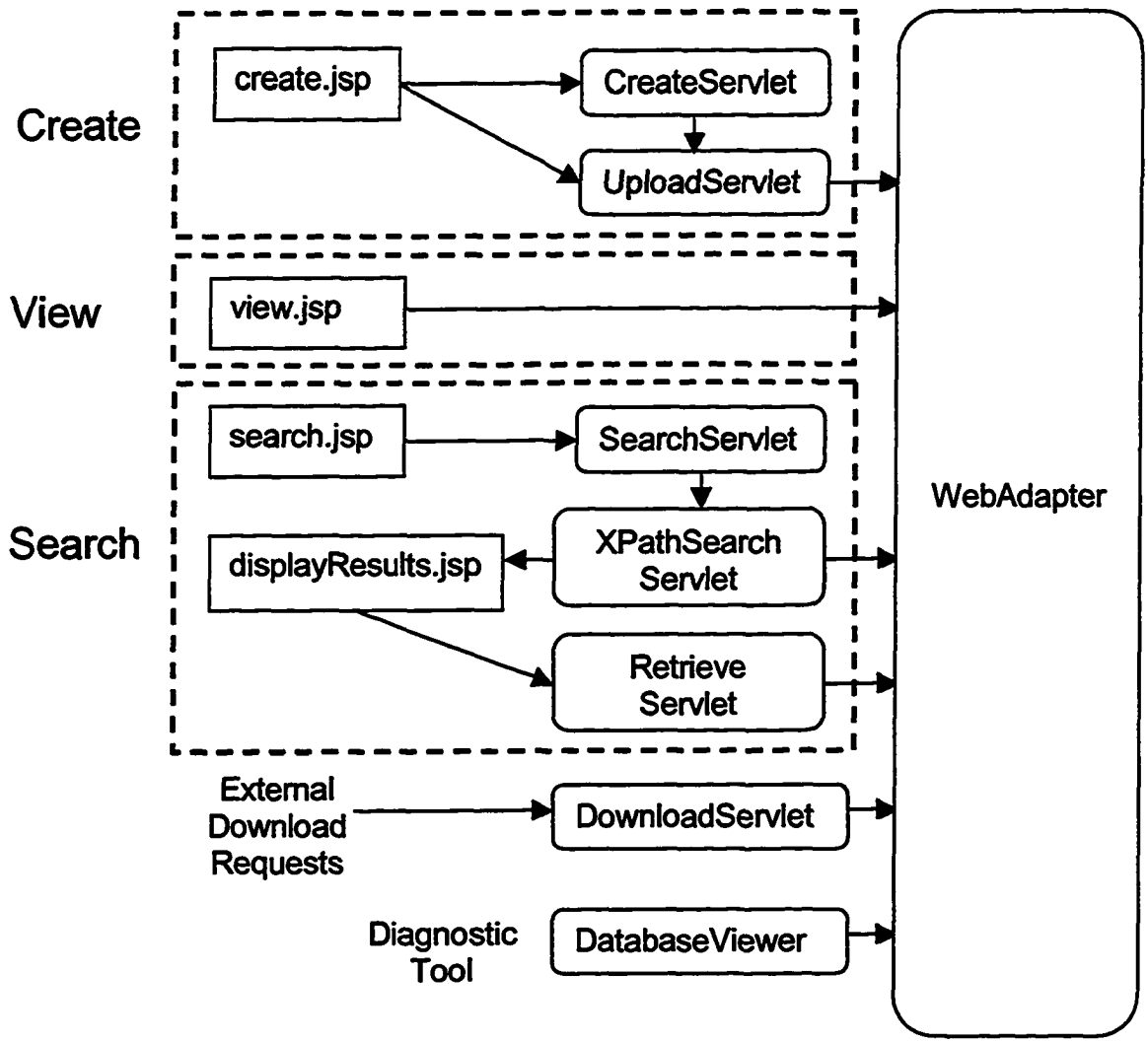

Figure 6 U-P2P JSPs and Servlets

3.2.6.1 Create

On the create page users will fill out an HTML form to manually create a resource or to upload a pre-made resource file. In the former case the XML is generated in the

CreateServlet and forwarded to the UploadServlet. The latter method of uploading a file goes directly to the UploadServlet. In the UploadServlet the file is parsed to check its conformance to the community XML Schema and feedback is given to the user for invalid files. If the resource conforms to the community schema, it will be published to the local XML database and to the peer-to-peer network via the community's Network Adapter.

To associate an attachment with a resource, the user will submit the file using the file upload facility of the web browser at the same time as the remainder of the form is submitted. For pre-made resources, an attachment is any part of the XML text that begins with file://, the URL for files. In either manual or pre-made uploads the UploadServlet will scan the incoming resource for attachments and store a reference to their location in a table.

3.2.6.2 Search

Searching for a resource in U-P2P is simple. The user selects the desired community, goes to the search page and enters search terms in an HTML form. As with the create page, the community founder can attach a custom search page for use with the community or in its absence, a default search page will be rendered by U-P2P. The fields that can be searched are dependent on the author of the search page, but generally they are all the same fields that can be entered on the create page, but without attachments.

Search terms in U-P2P are specified using XPath [Cla99] and consist of a set of XPath value pairs that must be matched in documents in the database or in documents shared by users on the network. For example, a search for a book might use the XPath value pairs

`/book/title=Great Expectations` and `/book/author=Dickens`. These terms would be combined into a single XPath and sent to the local database and to the peer-to-peer network. If the underlying peer-to-peer network does not accept custom search messages such as XPath value pairs, then the Network Adapter is responsible for translating the XPaths into a compatible query.

3.2.6.3 View

Viewing a resource takes advantage of their XML formatting to render them in a web browser. The community founder will attach an XSLT stylesheet that transforms XML into HTML or in the absence of such a stylesheet, a default stylesheet included with U-P2P will attempt to render a view of the resource. Due to the flexible nature of XSLT stylesheets, the community can view their objects with any format that is web browser compatible. For example, a community that shares chemical molecules could render them in a Java applet that presents a three-dimensional, interactive view of the object.

On the view page the user can also browse through the resources shared in a community and stop sharing (or remove) resources. Resources will only appear in a community once they have been shared (using the Create page) or downloaded from a search result.

3.3 Conclusion

U-P2P is a framework for sharing XML data in a community of users under the same schema. The shared data can be metadata describing a binary attachment or it can be any other data in XML format that needs to be shared.

Having covered the background of U-P2P and its architecture, the following section will outline why it is suitable for peer-to-peer data integration.

Chapter 4 Data Integration with U-P2P

U-P2P has several advantages over existing platforms that make it a strong candidate for using in peer-to-peer data integration. These advantages make U-P2P well suited to integrate data in a dynamic networked environment with many data sources and users.

4.1 Extensible Peer-to-Peer

U-P2P allows any user to create a new community of interest based on an XML Schema and network adapter. This is the equivalent of giving every user the power to create a network like Napster or Gnutella, but centered on their particular type of shared data. This is important for data integration because it allows networks of data to be formed around existing XML Schemas. Since hundreds of XML Schemas exist for many different languages and topics, a data integration system will greatly benefit by using the existing schemas and XML instance data.

4.2 Rich Metadata Searches

U-P2P goes beyond simple keyword searches used in file-sharing systems and allows structured metadata to be published and searched on the network. This allows data to retain a measure of semantics by keeping its structure and data types intact. Traditional file-sharing systems use keywords, file names and attributes extracted from selected file types but do not provide enough information for complex searches. Complex searches are required to search across integrated data as the structure of the data may hold semantic information that is relevant to mappings between schemas.

4.3 Flexible Peer Deployment

Through the Network Adapter modules, U-P2P also allows flexibility in how a community is deployed. New protocols can be incorporated by writing a new adapter and plugging it into U-P2P. This flexibility means data sources can access the network with the latest protocols and not be tied to one topology or type of deployment. Certain network deployments may also be unsuitable for peers sharing large amounts of data in an integrated system. For example, a protocol that requires uploading all metadata to a centralized index would not be suitable for peers with large databases.

4.4 Drawbacks and Alternatives

A disadvantage of using U-P2P for data integration is the lack of a proven track record for deploying U-P2P. So far U-P2P has not been used in a wide-scale deployment with large data sets and may need further improvements in stability and performance.

An alternative framework for consideration in peer-to-peer data integration is JXTA [JXT05]. JXTA is a set of open protocols that can be used to create peer-to-peer applications and services. It includes protocols for peer discovery, traversal of firewalls through rendezvous peers and protocols for establishing pipes between peer endpoints. JXTA provides the building blocks for a peer-to-peer application, but it does not provide the infrastructure available in U-P2P. Working at the protocol level, JXTA could provide an implementation for a Network Adapter in U-P2P. However, implementing a Network Adapter for JXTA was not needed to allow the framework to perform peer-to-peer data integration. Similar to JXTA, other file-sharing peer-to-peer protocols such as those used by Kazaa [Kaz05] and the distributed hash-tables of Chord [Stoica03] could be used as a

basis for data integration. However, these protocols are focused on providing complete queries and scalable topologies and a framework would have to be built on top of them for data integration. Instead U-P2P offers a framework where advanced protocols can be used while providing a basic set of functionality across all protocols and topologies.

To allow U-P2P to act as a data integration system will require some changes and improvements to the framework. The following section outlines how U-P2P will provide peer-to-peer data integration and the changes required to make it work.

4.5 Requirements for Data Integration Using U-P2P

U-P2P in its initial version does not have any means to relate one community to another on a schema-level or a semantic level. As such, data integration is not possible without making key changes to the framework and improving support for new data sources and network topologies.

First, a Network Adapter implementing a fully-distributed peer-to-peer protocol is required to remove the dependency on a centralized server used for indexing metadata. A fully-distributed protocol scales up to a larger number of users which is essential for integrating large amounts of shared data.

A second requirement is to make large databases accessible on the peer-to-peer network using proxies. A peer acting as a proxy for a database translates incoming queries into the database query language and responds to data requests by translating data into the XML format used in a community. It effectively brings large amounts of data to the network without having to modify databases or harvest their metadata and translate it to XML. Providing a database proxy also requires using a distributed peer-to-peer protocol to

allow a live database to be connected to the network without having to send its metadata to a central server. This supplements the first requirement of providing a fully-distributed network adapter to scale U-P2P up to a larger number of data sources.

A third requirement is to create 'bridges' between communities. A user in one community should be able to query another community by translating their query to the different schema. This requires mappings between schemas and a mechanism to allow searches to be automatically translated to the target community. If enough mappings exist the query should be translated, dispatched to the target community and results should be returned in either the original schema or the target community's schema. A bridge in U-P2P also needs to describe the semantic relationship between communities to allow for higher level traversal of communities related within a domain. If a community uses an ontology described in OWL, the bridge should provide mappings in OWL to a target community. The bridge format needs to be flexible enough to support both kinds of mappings and new types of relationships between schemas that may be required in the future. U-P2P has to be modified to take advantage of the bridges and integrate their use into the search user interface.

4.5.1 Use Case Scenario for Using a Bridge

To further motivate the use of bridges in U-P2P, the following use case scenarios are presented as examples of when using a Bridge Community would be beneficial:

1. A user in the Carleton University Library community wishes to expand their search to include related resources in the University of Calgary library. The user

wants to search for books or journal articles using the schema of the Carleton library community and be presented with results in the same schema.

2. A research lab shares chemical molecules in a community in U-P2P. A student in the lab performs a search in the chemical community and results are returned with various chemical molecules and compounds. A link on the search results page offers to perform the same molecule search on a related medical database. The user follows the link and a search is performed in the medical database with the results presented in the medical database schema. The user was previously unaware of the existence of the medical database or its connections to chemical molecules.

The above scenarios are a small sampling of the possible uses of bridges in U-P2P and future software agents will be equipped to performs more complex tasks as envisioned in for the Semantic Web (see Tim Berners-Lee's vision in [Ber01]).

Use cases for an RDF query language in development by the W3C [Cla04] contain examples of using RDF queries in tourism, publishing and transportation. U-P2P makes use of RDF and OWL and should therefore be compatible with RDF query languages and their use in these and future applications.

4.5.2 Requirements for a U-P2P Bridge

Given the two simple scenarios above, the requirements for a bridge can be summarized as follows:

1. Provide a means to define a semantic relationship between two uniquely identified resources in U-P2P. The relationships may be a known property in

OWL/RDFS such as equivalence (see 2.4.4 for details on OWL) or a more complex user-defined relationship.

2. Provide a means to transform the instance data (i.e. the XML resources themselves) of one community to another community.

3. Provide a means to translate a query destined for one community into a query compatible with a different community.

When implementing the bridges, the information must be presented in a user-friendly way so that the above use cases can be realized. This requires a mechanism to quickly retrieve a list of bridges related to a community without having the user execute a manual search. Another scenario for presenting bridge information is to provide links for the user to run the same search in other communities through query translation. This requires that U-P2P lookup the mappings in all bridges related to the current community and offer links to perform the new searches. The use and presentation of the integration aspects of U-P2P will be largely dependent on the changes made to the framework and the mechanics of searching and displaying results.

4.6 Conclusion

This chapter presented the advantages of U-P2P for use in a peer-to-peer system of data integration and noted some of its weaknesses and drawbacks. While U-P2P is so far unproven on a large-scale deployment, it is a flexible framework that offers pluggable Network Adapters for different deployments and rich metadata searching suited to the structured and typed data provided as XML. Alternatives to U-P2P do not offer any

infrastructure for searching and sharing XML resources, but instead may be used as the underlying protocol for a Network Adapter.

U-P2P does not offer any means to integrate the schema that form the basis of communities. To enable data integration a series of requirements must be met that allow bridges to contain a set of mappings that can be used for data and query translation and a semantic relation that can be used in future work on semantically connecting U-P2P communities and data. These requirements will be addressed in the next chapter on using U-P2P to integrate data.

Chapter 5 Integrating Data with U-P2P

The requirements for using U-P2P in data integration lead to the modification of the framework in several areas and the development of new software for handling functions related to the Bridge Community. This included implementing a fully-distributed Network Adapter using the Gnutella protocol, allowing asynchronous searches, implementing a system for database proxies and implementing the Bride Community and schema. The Bridge Community provides a place for attaching mapping files used in a simple query translation mechanism implemented and used later in the case study. An OWL statement is also included in the bridge schema to allow for semantic data integration in future work.

5.1 Gnutella Network Adapter

The Gnutella protocol [Gnu05] is a well known fully-distributed protocol suitable for implementation in U-P2P. It is far less complex than JXTA [JXT05] or other hub and spoke or 'superpeer' protocols such as Kazaa [Kaz05] and is freely available in software libraries in several languages. The Gnutella Network Adapter in U-P2P was implemented using the Muse [Mus05] Gnutella library. Muse is a free (GPL) Java library that implements the message routing, connection maintenance and push file requests in Gnutella. It was used to route U-P2P searches over a Gnutella network using custom messages. Gnutella only provides a space in its query message format for file name searches so custom messages were required to make use of the fully structured and typed metadata available in U-P2P.

To implement the Gnutella Network Adapter, the Network Adapter interface in Figure 7 was implemented by a class called GnutellaAdapter.

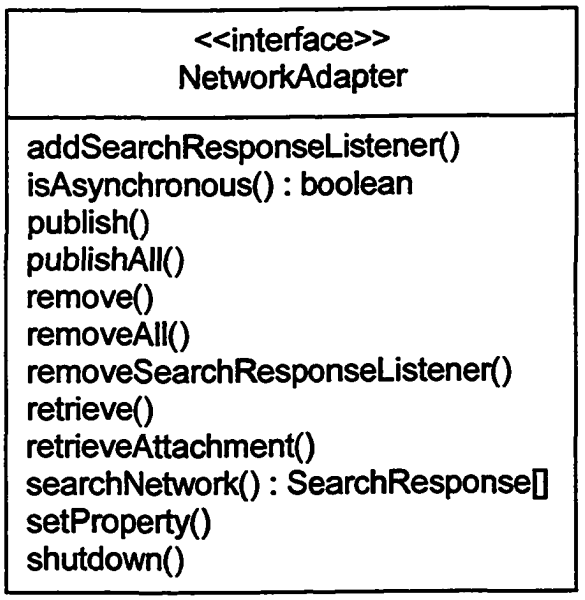

Figure 7 U-P2P Network Adapter interface.

Since the Gnutella protocol does not require peers to advertise their shared resources, the `publish()` and `remove()` methods were not implemented. This means that when the local user uploads or creates a new resource, the Gnutella adapter is informed through the `publish()` method (or the `publishAll()` method for batch processing), but does nothing to advertise the new resources on the network. Normally a Gnutella peer needs to maintain a list of shared resources but in U-P2P, requests for resources that come from other peers are handled by the DownloadServlet.

5.1.1 The Community Effect on Gnutella

The community aspect of U-P2P was also exploited in the Gnutella adapter by only using other community members as neighbours. This means that each community becomes a

small Gnutella network and grows as more peers join the community and share data. Figure 8 shows an example of the community effect on two communities using the Gnutella adapter. The peer in the overlap, *Peer 'A'* is a member of both communities but does not bridge the networks or relay any messages across community boundaries.

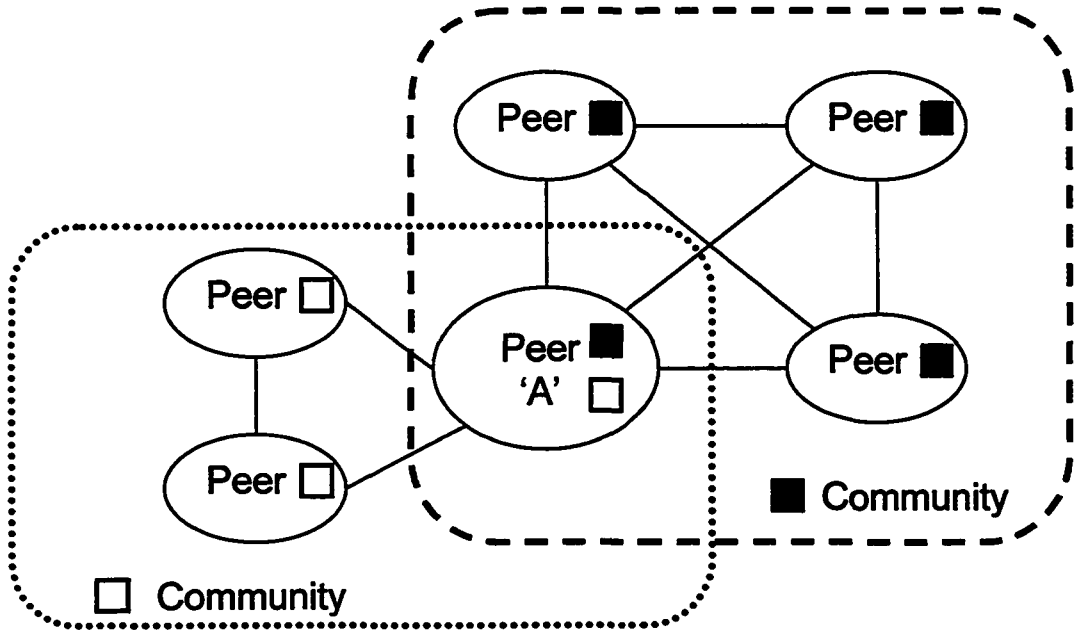

Figure 8 Two overlapping communities in U-P2P using the Gnutella Network Adapter.

The Gnutella methods of query flooding and message routing are used over a community of peers instead of a larger, possibly-disinterested network of peers. The grouping of peers into communities does not partition the network (as intentionally done in some P2P systems), because peers can be members of multiple communities each having their own overlay on the network of U-P2P peers. By only connecting to neighbours that are in the same community, it effectively creates a Gnutella network just for that community. A community-bound Gnutella network operates like any other Gnutella network so there is no advantage other than having a possibly smaller, more specialized group of like-

minded peers. A possible disadvantage is that there may be numerous small communities sharing very few resources.

5.1.2 Initial Host Caching

Since the Gnutella adapter is associated with a community and only connects to peers in the same community, the mechanism for bootstrapping the adapter is different from Gnutella peers on a public network. A community that uses the adapter will have its own cache file containing an initial list of hosts to which they should attempt a connection. Upon starting U-P2P this community-specific cache file is read and connections are opened up to other peers in the community. Unlike the public Gnutella network, there is no public web cache system such as GWebCache or deflectors that provide lists of known hosts on the network. This difference means that if all the hosts in the host cache for a community cannot be reached, the peer will not be able to connect to any other peers within that community.

To keep the list of cached hosts fresh, the host cache file is periodically refreshed with the current known hosts by the adapter along with the addresses of all its currently connected neighbours. This host cache file is an attachment to the community and therefore will be downloaded by any peer that joins the community. If a peer cannot connect to any hosts in their community host cache file then leaving the community and searching for it again in the Root Community may successfully find new community members. The community definition along with the host cache could then be downloaded again and the new host cache file used to connect to known members of the community. This system of host caching is a side effect of using a fully-distributed peer-to-peer protocol with no centralized list of users or a permanent known host.

5.1.3 Performing Asynchronous Searches

The `searchNetwork()` method in the NetworkAdapter interface dispatches queries to the network and in GnutellaAdapter, the implemented method forms a custom Gnutella search message and sends it out through the underlying layers of the Muse library. The search responses however, do not come back from the same method. The Gnutella network takes time to propagate the search and receive responses so the method dispatches the query and immediately returns. Responses coming in from the network are processed by Muse which calls `messageReceived()` in GnutellaAdapter. The GnutellaAdapter then processes the search results and sends them on to a listener that has been added to the adapter by calling `addSearchResponseListener()` as seen in Figure 7. The search response listener is the XPathSearchServlet and it caches the results in the user's session (a cache maintained by the Servlet container for persisting objects between requests). The page displaying results will periodically refresh itself and retrieve the new results from the cache. The results display page (`displayResults.jsp`) will only refresh if the `isAsychronous()` flag is true in the NetworkAdapter associated with the current search results. Figure 9 shows the structure of the classes involved in dispatching a search query and receiving an asynchronous response. Note that `messageReceived()` is only called when responses are returned from the Gnutella network which may not happen if the search is not fruitful.

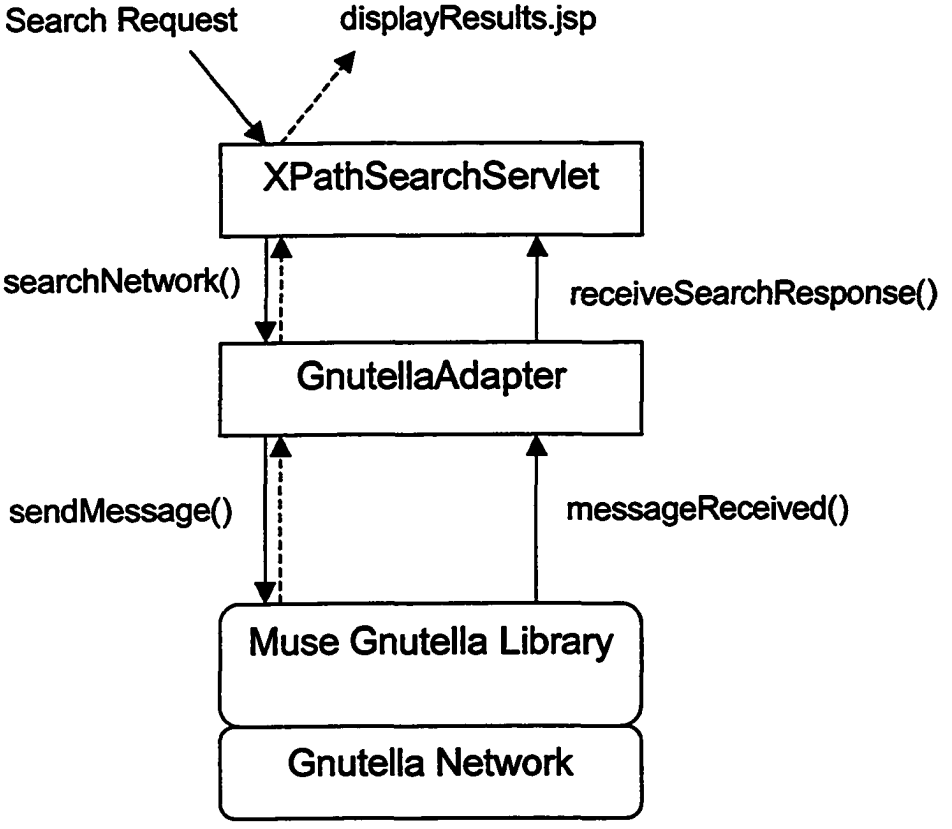

Figure 9 Messages exchanged in dispatching a query to the GnutellaAdapter

5.2 U-P2P as a Database Proxy

U-P2P was designed from the beginning with a 'Repository' interface to the backend database it uses for storing and indexing shared XML resources. This interface contains methods for searching the database, creating and deleting communities, and storing XML resources when they are published to the network. Figure 10 shows how the XML database is used by different parts of U-P2P. Using U-P2P as a proxy required replacing the module implementing the Repository interface with a module that connects to the live external database.

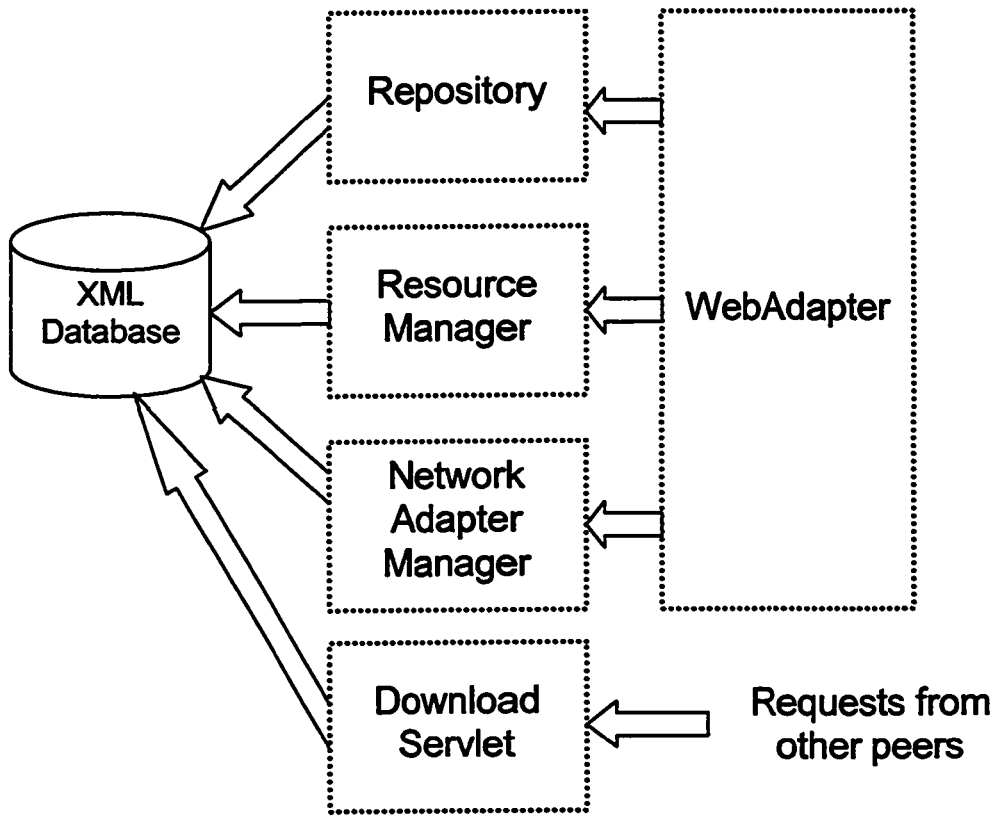

Figure 10 Flow of requests to the XML database in the unmodified U-P2P.

In addition to replacing the Repository implementation, requests for resources from other peers had to trigger a retrieval of data from the live database. In U-P2P requests from other peers are handled in the DownloadServlet. When a location is given for a resource in a search response, it uses an URL that points to the DownloadServlet on the peer hosting the resource. The DownloadServlet in turn uses a lookup table to find the original resource file and return it to the requesting peer. The lookup table (known as the FileMapper) is stored in the local XML database and this dependency is seen in Figure 10. To allow other peers to retrieve data from the live database, the proxy peer had to use an alternative implementation of the DownloadServlet to receive requests and translate data from the database format to the XML format of the appropriate community.

Solving the first problem of providing a new Repository implementation required changes to the framework to allow a provider for the Repository to be configured. Instead of loading the default Repository, U-P2P was modified to check a configuration file for the class name of the Repository provider. This is done when starting U-P2P so the user must decide beforehand if their peer will act as a proxy to a database. If the class of the provider can be found, the new implementation is loaded and all requests to the Repository proceed as in the original U-P2P. The implementation of the Repository is required to open and maintain connections to the live database, process U-P2P search queries (that are given as XPath value pairs) and translate responses from the database into the XML format used by the current community. The implementation may disregard requests to certain methods of the Repository interface, if for example it does not have the right to remove entries from the external database. This is important to allow a proxy to operate on behalf of a database that remains unmodified or even unaware of its participation in a data integration system. The proxy is free to implement any or all of the Repository interface methods and may use any protocols or libraries needed to contact the external database.

Since search requests in U-P2P use XPaths that are specific to one community schema, a proxy implementation may choose to only accept requests in one schema format and ignore all other requests. This means a proxy peer would typically only be a member of one community whose schema matches the relations in the external database.

The second step of providing a new DownloadServlet required modifying the framework to make a DownloadService interface to which provider implementations must conform. The original DownloadServlet then delegates all requests to the DownloadService

provider configured when U-P2P is started. This is the same arrangement as configuring a provider for the repository. Figure 11 shows U-P2P configured as a proxy to a live database with implementations of the Repository and DownloadService interfaces making connections to an external database.

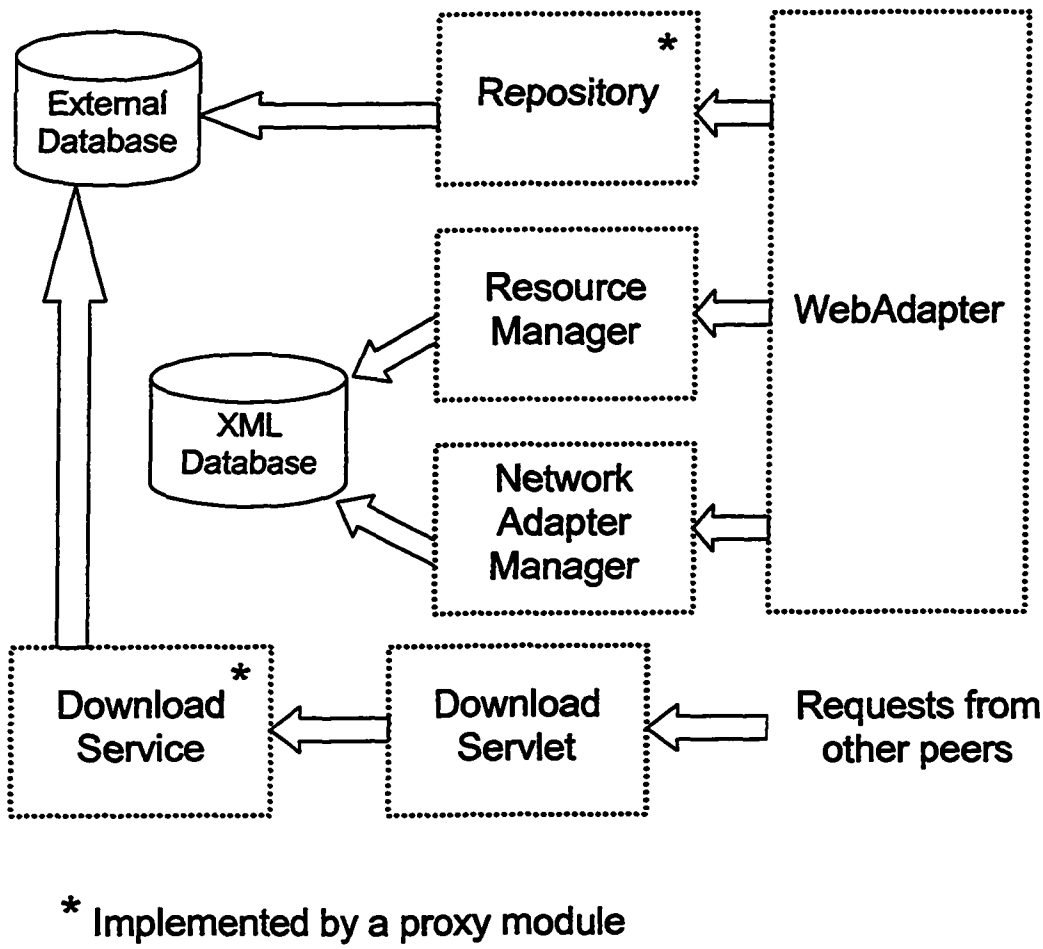

Figure 11 U-P2P configured as a proxy to an external database.

The details of implementing a database proxy are discussed in the case study in section 0.

5.3 U-P2P Bridge Community

The basis for a bridge in U-P2P arises from the need to provide an association between resources shared in different communities. If two schemas are authored that are in the same domain but, for example, are written for different audiences, it would be greatly beneficial if both communities could be searched without prior knowledge of both of their schemas. Thus a user could jump from one community to another because U-P2P knows about a relationship that exists between the communities and provides mechanisms for bridging the community schemas. The possible uses for a bridge are numerous and range from a user following a simple 'What's Related' link on a web page to more complex software agents interacting with multiple communities to traverse bridges and explore higher-level ontology mappings. For this thesis we will relate one community to another community using relationships such as equivalence and inheritance and provide a means to enter user-defined relationships of greater complexity.

To create a bridge in U-P2P the components of a bridge are collected together and published as a resource in the Bridge Community. Just as communities are published in the Root Community, bridges will be published in the Bridge Community where other peers can find the bridge and retrieve it for use in query translation and ontology mappings.

One key point of building a Bridge Community in U-P2P is to decouple the making of bridges from the publishing of communities. Anyone can publish a community in the U-P2P Root Community and anyone can create a bridge in the U-P2P Bridge Community. This means that a user who creates a bridge may not be the author of either of the communities that the bridge connects, but instead, may be a third party who is simply

making the connection and publishing it for others to use. This is analogous to the World-Wide-Web where everyone is free to comment on and link to each other's web pages and create their own content.

The following section outlines how the U-P2P bridges were implemented with details of the Bridge Community and its schema to follow.

5.3.1 Building a U-P2P Bridge

To fulfill the first requirement in section 4.5.2 of providing a semantic-level relationship, a U-P2P bridge will contain an OWL statement that relates one resource to another resource. This provides a simple high-level statement about how the two communities are related, without getting into a complex ontology. An OWL document can be attached for use in describing in greater detail how the resource and/or its components relate to an ontology.

Each resource will be identified by a Uniform Resource Identifier (URI) and the relation statement will be OWL/RDFS or a user-defined XML format. Since communities in U-P2P are themselves resources, one community can be related to another community by relating two resources in the Root Community.

It is important to note that the second and third requirements for a bridge, data and query translation, are specific to the case when both resources are shared in the Root Community and thus represent communities themselves. They are not applicable when both resources are instance data outside of the Root Community. Applying a translation to the schema of a resource allows all resources following the schema to also be translated.

The requirement of translating instance data will be satisfied by providing a place in a U-P2P bridge to affix a document or file that is used to transform data. Since each relationship is different we do not restrict the type of document used for the transformation and leave it up to the author of the bridge to attach a transformation file that is suitable for the relation defined in the bridge. As seen in section 2.4.6 schema-level mappings can be complex and require their own languages and algorithms for processing. The processing of these mappings may be performed by a piece of software that is specific to the relation and could range from an XSLT or XQuery engine to a custom module that is intimately familiar with both communities it is bridging.

The third requirement for a bridge of providing query translation is specific to the relation defined in the bridge. The same mappings used to translate instance data may also be used to translate queries. Or the query and data translations may require separate mappings or transforms. The bridge schema is flexible and supports any number of such mappings in any format to accommodate new types of mappings.

Having outlined what should be in a bridge, the schema for bridges is presented in the next section followed by details on the use of OWL in the bridge.

5.3.2 U-P2P Bridge Community Schema

The complete bridge XML Schema is available in Appendix A and can be described as follows (see the XML Schema for the syntax and structure of the XML elements):

- Metadata – Title and description of the bridge. What the bridge does and what it is used for in general terms.
- Bridge Source – ID of the resource and community of the source of the bridge.

- Relation – OWL relationship between the two resources. This can be replaced by user-defined relationships using an XML namespace other than OWL.
- Bridge Target – ID of the resource and community of the endpoint of the bridge to which the source relates.
- Transform List – List of binary attachments with an accompanying `type` and `description` that are used for translating queries and data.

The nature of the data transformation or the query translation depends on the relationship between the resources and therefore any type of file may be included in the Transform List section. The use of transform files is dependent on the implementation of the query translation or data transformation.

The bridge can be as complex as necessary and allows any number of attachments containing mappings that be used in data and query translation. The mapping or transform attachments are accompanied by a 'type' and 'description' that can be used by U-P2P to identify a type of transform and execute the appropriate behaviour. The caveat is that U-P2P has to know what to do with the mappings and how to use them. For the case study in section 0 a simple XPathMap is used for query translation and translation of instance data is not implemented.

5.3.3 XPath Query Translation

For the case study a simple XPath query translation was used to translate queries from one schema to another. XPath mappings are expressed in a proprietary XML format and attached to the bridge between two communities.

The following is an example of an XPath mapping:

```
<XPathMappings>
  <mapping source="/item/metadata/dc:title"
    target="/record/name" />
</XPathMappings>
```

In this example the XPath `/item/metadata/dc:title` is mapped to `/record/name`. It is important to note that the target XPath can be more complex but that overall, the mappings are a simple translation mechanism that can be supplanted by more complex data translation schemes. With this method of query translation the data is not typed and therefore cannot be manipulated as in a full query language such as SQL.

When a request is submitted to the XPathSearchServlet to translate a query, it finds the XPath mappings and performs a lookup of each XPath in the search query. If a match is found in the source of the mapping, the target XPath is inserted in its place. If no mapping is found, the XPath search term is omitted from the translated query.

After query translation, the translated query is processed like any normal query and sent both to the local repository and to the network through the NetworkAdapter. When displaying the search results after performing a search in a community, a link is displayed to allow the user to perform the same search in a related community. By clicking this link the query translation in the XPathSearchServlet is called with the ID of the bridge to be used to perform the translation. The XPathSearchServlet retrieves the current search from the user's session information (as previously stored in the initial search) and the bridge information from the local XML database and performs the translation using a method in XPathQueryTranslation, a new class written for translating XPath queries.

5.3.4 OWL Bridges the Gap

In a U-P2P bridge, an OWL/RDFS statement defines the relationship between a source and target resource. If the resources are in the Root Community, then the bridge defines a relationship between two community schemas. Otherwise, the relationship is between resources and may not require mappings for the translation of data and queries. In either case, the statement is written in OWL and since OWL includes RDF and RDF Schema (RDFS), it may include RDF/RDFS statements.

As seen in section 2.4.2, an RDF or OWL statement is a property that relates an *object* to a *subject* using a *predicate*. The object and subject are both U-P2P resources that are implied in the bridge mapping (identified respectively as source and target). The predicate is either a OWL/RDFS predicate or a user-defined predicate in a separate namespace.

OWL includes the following properties that may be useful in creating a bridge between two resources:

- `rdfs:subClassOf` – resource A is a type of resource B
- `owl:disjointWith` – resource A excludes all of resource B
- `owl:intersectionOf` – resource A is the intersection of resource B and C
- `owl:unionOf` – resource A is the union of resource B and C
- `owl:complementOf` – resource A is the opposite of resource B
- `owl:sameAs` – resource A is the same as resource B

- `owl:differentFrom` – resource A is not the same as resource B (despite any similarity)

Note that the properties involving a third resource (resource C) are more difficult to implement in data translation and query translation. For the case study in section 0, only the `owl:sameAs` relationship is used with simple query translation.

Since OWL relates one resource to another, U-P2P needed to have a format for uniquely identifying resources. Although the original U-P2P uses unique hash values for each resource, a URI format will serve to identify a resource and its context community.

5.3.5 U-P2P URIs

A URI is a unique name attached to an object that resembles a Uniform Resource Locator (URL) (e.g. http://www.example.com/item). Unlike an URL, a URI does not imply that an object resides at the location it describes.

The U-P2P URI format is as follows:

up2p:community=<community id>|resource=<resource id>

Where <community id> and <resource id> are unique hash values assigned to the resources when they are uploaded to U-P2P.

To identify an attachment in U-P2P, the URI format is extended to:

up2p:community=<community id>|resource=<resource id>|attachName=<attachment name>

The attachment name (<attachment name> in the URI) is the file name of the attachment

associated with the U-P2P resource. Attachment names must be unique within the scope of a resource and only contain a file name with no further path information.

The U-P2P URI format can be used in OWL/RDF statements for semantic-level mappings.

5.4 Conclusion

This chapter detailed the bridge schema and Bridge Community created in U-P2P to integrate data. The bridge contains mappings used by translation modules to translate queries and instance data (with the former being implemented for the case study) and an OWL statement for a semantic mapping. A Network Adapter using the Gnutella protocol was implemented to allow a fully-distributed system of data integration. The mechanisms for using proxies were also introduced to allow U-P2P peers to act as proxies for large databases.

Chapter 6 Digital Repositories Case Study

The effectiveness of the U-P2P framework and its extensions for integrating data are best demonstrated by using the framework on a case study scenario. A candidate for peer-to-peer data integration is in digital libraries or repositories. With the lower cost of online storage and the availability of metadata protocols, institutions in both the public and private sectors are taking a more active role in the preservation of digital work products [Lyn03]. This has led to the development of digital repositories that store and index electronic documents in a permanent archive for preservation and dissemination with generally open but user-based access. In July 2004, the British House of Commons Science and Technology Committee published a report on the state of scientific publications in the United Kingdom [HCS04]. It recommended that all higher education institutions establish repositories and make their output available for free online. The Commons points out that if "all countries archived their research findings in [institutional repositories], access to scientific publications would increase dramatically" [HCS04]. With institutional repositories on the rise there is a clear need to integrate data from multiple repositories.

By their nature, archives gather data into one place to facilitate access control and maintenance. This leads to each institution having its own digital archive running repository software that may be custom-built, commercially purchased or an open-source package. With many different repositories all running their own software or even different versions of the same software, searches spanning across archives are not possible without custom applications that glue the archives together. For example, a user

searching the repository at MIT might want to search for similar documents on the same subject published at Berkeley. Since each university runs different repository software (DSpace and BEPress respectively), an application or user would have to manually perform each search, retrieve the results and then combine them for a meaningful result.

6.1 OAI Harvesting

Some digital repositories support a form of data integration using the OAI Protocol for Metadata Harvesting (OAI-PMH) [Lag02]. The Open Archives Initiative (OAI) is an organization that promotes standards for the "efficient dissemination of content" [OAI05]. The OAI-PMH is a protocol that allows software scripts to harvest metadata from OAI-compatible archives. Any types of XML metadata can be harvested, but repositories typically only support terms from the Dublin Core Metadata Initiative [DCT05] such as 'title', 'creator' and 'date'.

OAI-PMH is used by harvesters to periodically poll a data source and retrieve all its new metadata records. It is not a search protocol and only provides retrieval and listing of metadata records that are relatively small in size. This is analogous to allowing members of the public to enter a library, copy the complete library catalogue from the computer and then take the copy home to peruse. Typically the harvested metadata is assembled into a larger collection and indexed by a search engine that is made accessible to the public (e.g. OAIster [Ost05]). Such aggregators of OAI metadata provide gateways to several collections but suffer from constantly having to harvest their metadata records to keep them up to date. They also cannot take advantage of any metadata not supported by the harvester, such as non-DCMA metadata and they usually only target the largest OAI-compatible archives for harvesting. The OAI protocol is supported by both repositories in

the case study for the Dublin Core metadata terms. While its use in harvesting metadata from large repositories is proving useful in aggregators, the OAI-PMH only scratches the surface of data integration and is not suited to a dynamic environment of numerous changing data sources with rich semantic metadata.

6.2 Repository Deployment Scenario

To demonstrate the flexibility of U-P2P in its capacity for data integration, a deployment scenario is presented below to suggest how two digital repositories might be used and integrated in U-P2P. The two digital repositories used in this case study, DSpace and Fedora will each have their own community in U-P2P. The DSpace and Fedora communities are representative of how information from these digital repositories can be shared on a peer-to-peer network and integrated using bridges. More specialized versions of the communities could be used instead to allow data integration between portions of a repository (e.g. a department in a university) or customized versions of the DSpace and Fedora metadata.

For this case study, three peers will be deployed that span two types of networks. In Figure 12 Peer A and Peer C are members of the DSpace and Fedora communities (whose schemas are presented below). Peer B is a member of Fedora Community and is a proxy for a Fedora database. The Fedora Community uses the Gnutella peer-to-peer protocol implemented in the GnutellaAdapter class and the DSpace Community uses the generic centralized (Napster-style) adapter implemented in GenericCentralPeerToPeer.

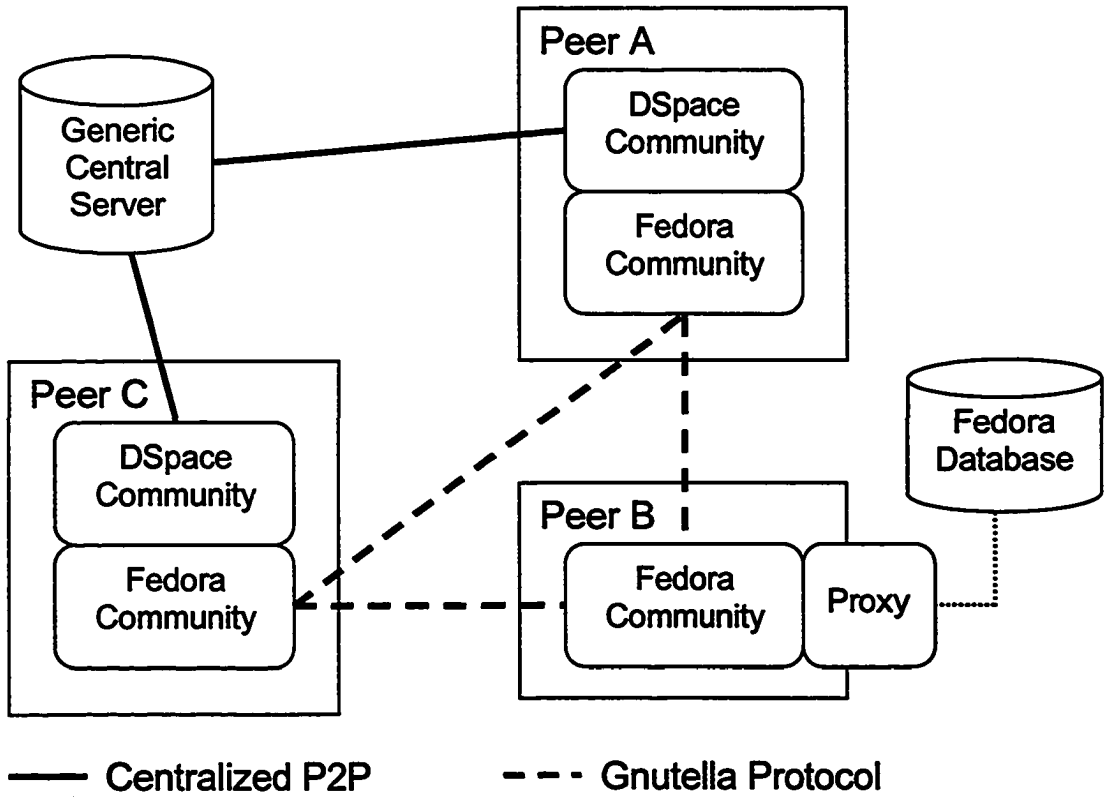

Figure 12 Case study peer deployment.

6.2.1 Search in the Fedora Community

If Peer A or Peer C performs a search in the Fedora Community, it will dispatch a Gnutella query message to its neighbours and receive responses as described in 5.1.3. Peer B is a proxy peer for a Fedora database and will translate the query into the Fedora format and query the database directly. It will then form a query response message and return it to the requesting peer.

6.2.2 Search in the DSpace Community

A search in the DSpace community is sent from Peer A or Peer C to the centralized server that holds an index of all shared resources. The server will response to the query

and direct the peer to the appropriate download location. The topology of the centralized peer-to-peer model does not require Peer A to have any knowledge of Peer C and the connection in Figure 12 only applies to the Fedora Community which has a different deployment.

6.2.3 Integrating a Search across Both Communities

Having shown how each peer searches in their network, we now want to perform an integrated search across both communities. For example, Peer A wishes to take a search performed in the DSpace Community and execute it in the Fedora Community. To perform an integrated search across the communities a bridge is required to provide a mapping to translate the query. If we assume such a bridge exists and is shared in the Bridge Community (deployed for example, using the Napster-style topology), then Peer A needs to search for and download the bridge before a translated search can take place. A future version of U-P2P may automate the discovery of bridges to help make integration smoother and easier to use. For now, we assume Peer A finds and downloads the bridge and performs a search in the DSpace Community. On the results page for the query, a link is displayed allowing the user to run the search in the Fedora Community. By following this link, query translation is triggered as described in section 5.3.3 and the translated query is run in the Fedora Community. By running the query in the Fedora Community, Peer A dispatches a search request to Peers B and C using the Gnutella protocol. The proxy Peer B would in turn query its Fedora database and return any matching results. The responses from Peers B and C are displayed to the user in the Fedora Community and the user has effectively changed communities without entering a new query that uses the Fedora Community schema. Thus the query translation saves the

user from manually translating searches in communities that are related through bridges. The flexibility of U-P2P in connecting to multiple peer-to-peer networks in different configurations allows bridges to be built across networks of peers that may include proxies to live databases.

6.3 DSpace

DSpace is a system developed by the MIT Libraries and Hewlett-Packard Labs, and its main focus is storing digital output created by faculty and staff of a research institution [Dsp05]. It aims to capture, index and provide access for the redistribution of documents, media and data produced by a university. This type of repository is often called an institutional repository or IR. Although DSpace is oriented toward scholarly work products from a university and incorporates mechanism for workflow management, we will be treating it simply as a repository for documents and media.

6.3.1 DSpace Object Model

DSpace is oriented around a stack of containers or objects that range from the top level 'Community' to the bottom-level 'Bitstream'. The hierarchy of DSpace objects is shown in Figure 13. The topmost object, a Community, is a starting point for referring to a group of collections and typically corresponds to a university faculty or department. A Collection is a grouping of related DSpace items and an individual Item is a single "Archival Atom" used in DSpace. This means users store, retrieve and search for Items and not the lower-level Bitstreams. Bitstreams are a sequence of bits that have a specific format (i.e. MIME type) and are associated with an Item. An Item can have multiple Bitstreams which are bundled together whenever an Item is stored or retrieved.

Bitstreams may contain the content of an Item (such as an electronic text) or metadata about the item in a form specific to the Item. Metadata in a Bitstream is not searchable but instead, the metadata entered when submitting the object is used for indexing.

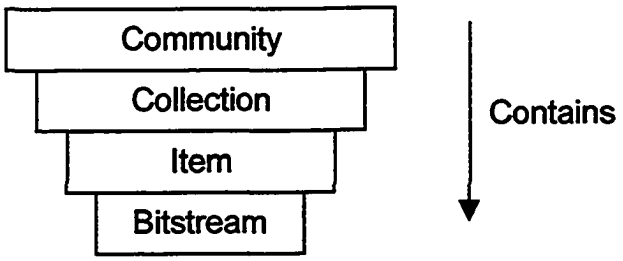

Figure 13 DSpace hierarchy of objects.

6.3.2 DSpace Metadata

Metadata in DSpace is based on the Dublin Core Metadata Initiative Library Application Profile [DC-Lib04] and contains Dublin Core elements as well as refinements specific to DSpace. The Dublin Core consists of fifteen core elements and a number of refinements that qualify them. For example, the 'title' element can be refined by the 'alternative' qualifier to specify an alternative title for a work.

The Library Application Profile was developed for use by libraries and specifies a set of elements and qualifiers that are permitted for use along with a set of data encodings for certain data types. The Dublin Core specifies data encoding for items such as Library of Congress Classification numbers and RFC 3066 language and country codes.

Metadata is entered in the DSpace system when an item is submitted and is stored in a relational database that also indexes and provides search capabilities for the metadata. The metadata is a fixed set of terms that cannot be expanded to include more metadata from the Dublin Core or other sources, unless the DSpace application and database are

modified accordingly. The current implementation sets up tables in the relational database with fields for each of the metadata elements that are supported by DSpace and when necessary, the appropriate data type.

Although not used in DSpace, the Dublin Core does have guidelines for implementing the Dublin Core elements in XML and these were followed, as detailed below, when DSpace was adapted to U-P2P.

6.3.3 DSpace Interface

DSpace is written as a web application that is deployed on the server controlled by the repository administrators and is backed by a relational database running on the same machine or in another location. A centralized model is employed to allow for access control and security and to collect the digital resources in one physical location for the purpose of creating a permanent archive. Much like a library maintains a central database of its contents, DSpace would be run by the library and staff and faculty would access it through its website. For example, the DSpace repository at MIT is publicly accessible at http://dspace.mit.edu. Access control is provided by user accounts set up on the web server and secure connections are maintained using a Secure Socket Layer (SSL) connection.

6.3.3.1 Submitting an Item to DSpace

A typical user session for storing a new object in DSpace might be as follows:

1. Collect the digital objects or documents and prepare them for archiving.

2. Login to the DSpace repository website of the institution using an account setup with the appropriate access rights. DSpace authorizes groups of users for each Community, Collection or Item.
3. Browse to the appropriate Community and Collection and follow the links to submit a new document.
4. Enter metadata in the HTML form with the 'title' being the only required element. Some metadata is populated automatically be DSpace (such as the permanent URI handle and accessioned date).
5. Confirm metadata validity and edit the metadata if necessary.
6. Upload Bitstreams and described them (if uploading more than one).
7. Finally, confirm the Bitstreams and descriptions and complete the submission. A workflow may be in place for another user to review the submission before it gets posted to the public archive.

All steps performed on the DSpace website are implemented in Java Servelets and JavaServer Pages (JSPs) and deployed in Jakarta Tomcat. The DSpace application itself is an assembly of JSPs, Java objects, libraries and the necessary code to interface with the back-end relational database.

6.3.3.2 Searching in DSpace

Searching in DSpace has fewer steps than submitting an object and involves browsing to a Community or Collection and searching by keywords. An Advanced Search page allows Boolean searching of three fields from one of keyword, author, title, subject, abstract, series, sponsor or identifier. Each of these terms corresponds to a Dublin Core

metadata element and includes a search on refinements of each element. Searching and indexing of text is handled by the Jakarta Lucene open-source text search engine [Luc05]. When Items are submitted and approved for entry in the system their metadata is indexed by Lucene and stored in the relational database along with information on the Bitstreams and a newly generated Item id.

6.3.4 DSpace Integration Features

DSpace was designed as a standalone archive of an institution's digital work product. It does not provide remote access to the search facilities of DSpace other than through the website of the repository host.

DSpace, with the help of included Java Servlets, does provide a service for the OAI Protocol for Metadata Harvesting [Lag02]. Only Dublin Core metadata is exported to the harvester and metadata such as time of submission, collection id, bitstream descriptions and MIME types. Other than OAI harvesting, DSpace has no externally visible APIs for accessing its search facilities or specifically for integration. DSpace provides tools for exporting the contents of a DSpace repository to a simple XML format or to the Metadata Object Description Schema (MODS), a Library of Congress standard [MOD05]. While exporting records to XML could be useful in integration, it can only be performed by the administrators of the database and is intended for backing up data.

6.3.5 U-P2P and DSpace

To integrate DSpace with another repository we first need to emulate DSpace with U-P2P. This will allow DSpace records (and their bitstreams) to be imported into U-P2P as XML metadata and attachments. Despite the complexity of the DSpace application and

its robust feature-set, the main functionality of DSpace was recreated in U-P2P in a matter of hours. The following steps need to be performed to create a DSpace U-P2P community that emulates the storage, retrieval and searching capabilities of DSpace:

1. Create a DSpace schema that will store the metadata and Bitstream information used in DSpace. The XML encodings for Dublin Core elements suggested in [Pow03] were used when possible and additional elements for DSpace data were added when needed. DSpace uses some non-standard qualifiers for DC terms and these had to be added to the schema. A section of the schema defines a place for the Bitstreams to be listed as attachments in U-P2P with their corresponding descriptions, sizes and dates.
2. Create the DSpace XSLT display stylesheet to display resources in 'full' and 'simple' record modes (as seen in the DSpace interface).
3. Create the DSpace community HTML Create page for U-P2P.
4. Create the DSpace community HTML Search page for U-P2P.
5. Copy the DSpace CSS stylesheets and adjust them to suite the view, search and create pages.
6. Create the DSpace community schema file by describing the community and listing the above files.
7. Start up U-P2P and publish the newly created DSpace community.

Of these steps, writing an XSLT stylesheet and reconciling the Dublin Core metadata terms used in DSpace consumed the most time.

The following is an example DSpace record showing the Dublin Core metadata and a list of attached Bitstreams (or attachments as they are called in U-P2P):

```
<?xml version="1.0" encoding="UTF-8"?>
<dspace xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns:dcterms="http://purl.org/dc/terms/">
  <item>
    <metadata>
      <author>Saint Augustine, Bishop of Hippo</author>
      <accessionedDate>2005-03-24</accessionedDate>
      <dcterms:issued>0397</dcterms:issued>
      <isbn>xyz</isbn>
      <identifierURI>http://foo</identifierURI>
      <dc:description xml:lang="en">English translation of Augustine's
Confessions, Book Ten</dc:description>
      <dcterms:abstract xml:lang="la">Magnus es, domine, et
laudabilis ualde: magna uirtus tua et sapientiae tuae non est numerus.
Et laudare te uult homo, aliqua portio creaturae tuae, et homo
circumferens mortalitatem suam, circumferens testimonium peccati sui et
testimonium, quia superbis resistis: et tamen laudare te uult homo,
aliqua portio creaturae tuae.</dcterms:abstract>
      <dcterms:extent>23 pages</dcterms:extent>
      <mimeType>application/pdf</mimeType>
      <isoLanguage>en</isoLanguage>
      <isPartOfSeries>Book X of XII</isPartOfSeries>
      <dc:subject>philosophy</dc:subject>
      <dc:subject>early Christian literature</dc:subject>
      <dc:subject>fathers of the Church</dc:subject>
      <dc:subject>memory</dc:subject>
      <dc:subject>conscience</dc:subject>
      <dc:subject>religion</dc:subject>
      <lcc>BR65.A6C6 1931  v. 1</lcc>
      <dc:title>Confessiones</dc:title>
      <dcterms:alternative>St. Augustine's
confessions</dcterms:alternative>
      <dcterms:alternative>Confessiones. English &amp;amp;
Latin</dcterms:alternative>
      <dc:type>Book</dc:type>
    </metadata>
    <collectionList>
      <collection handle="http://localhost:8080/up2p/...">Early
Christian Philosophy Texts</collection>
    </collectionList>
    <bitstreamList>
      <bitstream>
        <name>augustine-confessions.pdf</name>
        <description>Book Ten, English text translated by Albert C.
Outler, Ph. D.</description>
        <size>83700</size>
        <mimeType>application/pdf</mimeType>
        <formatDescription>Adobe PDF</formatDescription>
        <location>file://test/dspace/augustine-
confessions.pdf</location>
      </bitstream>
```

```
    </bitstreamList>
  </item>
</dspace>
```

In the above sample item, Dublin Core terms with a namespace prefix of 'dc' or 'dcterms' are referencing Dublin Core elements that are defined by the DCMI in an XML Schema document. Since DSpace uses additional elements that are not defined the DCMI schema, those without namespaces are defined in the community's XML Schema.

In some cases the Dublin Core specifies an encoding schema instead of a DC term. For example, LCC is specified as an encoding that can be used in the 'identifier' term (i.e. the identifier should have the LCC format). DSpace instead used the encoding schemes as qualifiers and thus qualified 'identifier' with 'lcc' to indicate its encoding. In terms of XML it was simpler to re-use any elements in the DCMI schema and then add the remaining terms as elements in the schema. For example, the term 'identifier' qualified as 'lcc' became the element <lcc> and can be seen in the above sample item.

6.3.5.1 U-P2P Modifications to Support DSpace

The DSpace community, as created using the above steps, would function in U-P2P if the user was to upload ready-made resources that conform to the DSpace community schema. However, to support manual creation of the DSpace resources through the create page of U-P2P, several features were added to U-P2P to insert information available at runtime. DSpace generates metadata at runtime and inserts it into the record for a submitted item so U-P2P had to emulate this by using tags in the create page. Additionally, tags were created to facilitate the assembly of an ISO-formatted date that can be inserted at any given XPath in the created document.

In the create page of a community in U-P2P, each form field has a name and value in the underlying HTML code. As mentioned in the background on U-P2P, the create page functions by populating the form field names and value with XPath-value pairs. The name of the field is the XPath of the node that should be created in the target XML. The value entered by the user is inserted at the XPath in the name. For example, the following HTML creates a form with a text entry field:

```
<form action="create" method="post">
<p>Please enter the following information for your book:</p>
<p>Title: <input type="text" name="/book/title" size="30"></p>
</form>
```

The above HTML works for standard XPaths but does not allow the insertion of information determined at runtime. For example, DSpace lists the names of Bitstreams that are associated with an object. In U-P2P the attachments are only known after they are uploaded and therefore, runtime support is required to insert special values.

The added create tags include:

`up2p:attach-name-n` – Name of the nth attachment.

`up2p:attach-size-n` – Size in bytes of the nth attachment.

`up2p:attach-mimeType-n` – MIME type of the nth attachment as determined by file extension (e.g. .pdf is application/pdf).

`up2p:attach-mimeDesc-n` – Short description of the MIME type of the nth attachment (e.g. PDF File).

Also, dates in DSpace use the ISO standard formatting and in HTML this requires that the user enter the date in a text box in the non-intuitive format of YYYY-MM-DD (where the letters represent the numeric year, month and day). So as a courtesy to the author of

create forms in U-P2P (and not just in the DSpace community), support was added so the user could enter each part of the date separately and U-P2P would reassemble it and insert it at the desired XPath location.

Having created DSpace in U-P2P as a community, it can now be integrated with another digital repository community.

6.4 Fedora Digital Repository

Fedora is a digital repository framework developed jointly by the University of Virginia and Cornell University [Sta03]. It stores digital objects and defines mechanisms to disseminate the digital objects using various services for rendering the content. The system de-couples the content from its delivery and uses a somewhat complicated scheme of behaviour definitions and behaviour mechanisms to define object disseminators.

6.4.1 Fedora Object Model

There are three kinds of objects in Fedora: Data Objects, Behaviour Definition Objects and Behaviour Mechanism Objects.

Data Object

Data Objects contain the actual content such as a document or digital image. They have a unique PID and one or more Datastreams that are analogous to DSpace Bitstreams (and U-P2P attachments). A Data Object is also associated with one or more Disseminators.

Behaviour Definition

A Behaviour Definition Object, when combined with a Behaviour Mechanism Object forms a Disseminator. A Disseminator presents the data in the repository, for example by

transforming it into HTML using an XSL stylesheet. A Behaviour Definition defines the public interface of a Disseminator (i.e. what methods can be called on the Disseminator). Data in the repository associates itself or 'subscribes' to a Disseminator by associating with the PID of the Disseminator's Behaviour Definition.

Behaviour Mechanism

The Mechanism object is the concrete implementation of the interface defined in the Definition object and can provide methods suitable for viewing and manipulating specific types of data. For example, Fedora includes a demonstration object that uses an image-scaling disseminator with a Behaviour with methods for retrieving High, Medium, and Low resolution images from the same digital image source.

All three types of objects in Fedora are managed in the repository as Data Objects and clients can search and find Behaviour objects as well as regular data. This is similar to U-P2P's ability to publish a community in the Root Community to allow other users to find it and join the community.

6.4.2 Fedora and METS

All three object types are stored as XML in the Library of Congress Metadata and Encoding Transmission Standard (METS) format [METS, 2004]. METS follows the above data model closely and has sections for descriptive and administrative metadata, attached digital files, and descriptions of behaviours to be executed on the content of the METS record. Although the data in the Fedora repository is stored in METS format, access to the data is performed through the APIs described below which do not return METS formatted data.

6.4.3 Fedora APIs and Access

Fedora defines two APIs: Repository Access Interface (API-A) and Repository Management Interface (API-M). Both interfaces are Web Services [Web05] with Web Services Description Language (WSDL) [Chr01] documents describing the methods and their contracts. By using Web Services, the behaviour mechanisms mentioned above can reside on any server on the accessible network, and thus not necessarily be on the same server as the data. This allows the behaviours associated with disseminators to be centrally located and maintained and serviced by its owner.

For this case study, a 'lite' version of the APIs was used that uses the same methods but accesses them through URLs on a web server. Fedora does not support all of the methods in the APIs through the web server and the management API in particular is lacking. This did not affect the case study, as we were able to access Fedora through default disseminators that return the data objects and their datastreams in XML and binary formats. The disadvantage of only using the lite APIs is that the URL syntax does not support submission of new objects to the repository and ingesting new data can only be done through scripts provided with the implementation and executed locally on the Fedora server. The URL syntax lite APIs return either HTML or XML depending on their parameters and the XML is in a custom format and not in the METS format described above.

6.4.3.1 Fedora Access API (API-A)

The Fedora Access API (API-A) provides search and retrieve methods for objects and disseminators available in the repository. A user can search for an object using Dublin

Core metadata that is associated with each object. The resulting permanent identifiers (PIDs), can then be used in a query to see all disseminators associated with a PID. Once the user finds the right disseminator, they call a method to get a dissemination, which Fedora then executes by calling the associated Behaviour Mechanism.

To facilitate exploring the repository, methods are provided in API-A that return a description of the repository or an Object Profile for any PID. The Object Profile contains a label, the owner ID of the user who created the object and other system metadata associated with the data object. Retrieving the object profile is a method of a default disseminator that is included with Fedora for the purpose of introspection and exploration of the repository. It also allows a client to download the WSDL of a Behaviour Definition and therefore learn of new disseminators.

6.4.3.2 Fedora Management API (API-M)

Fedora's other API is for management of Fedora objects. It contains methods for adding and retrieving raw datastreams to the repository, describing a user, exporting objects, and modifying objects and disseminators. As with API-A, all methods are defined in WSDL as a Web Service and are called from another application or front-end client.

6.4.4 U-P2P and Fedora

Fedora makes extensive use of web-oriented APIs that offer access to digital objects in an XML format. To share Fedora objects in U-P2P a schema was created that includes the system and descriptive metadata associated with a Fedora data object and links to the datastreams associated with an object. Although Fedora uses METS, an XML formatted

language, the access APIs for Fedora do not return METS records so a new schema was written that includes most of the METS record contents.

The following is a sample Fedora digital object (from the Fedora package) as seen in U-P2P:

```
<fedoraItem xmlns:dc="http://purl.org/dc/elements/1.1/">
    <!-- Object Profile -->
    <objectProfile pid="demo:5">
        <objLabel>Image of Coliseum in Rome</objLabel>
        <objContentModel>UVA_STD_IMAGE</objContentModel>
        <objCreateDate>2004-11-02T13:01:38</objCreateDate>
        <objLastModDate>2004-11-02T13:01:39</objLastModDate>
        <objType>Fedora Data Object</objType>
    </objectProfile>

      <!-- Data streams -->
    <objectItemIndex pid="demo:5">
        <item>
            <itemId>DS3</itemId>
            <itemLabel>Thorny's Coliseum high jpg image</itemLabel>
            <itemURL>http://localhost:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DS3</itemURL>
            <itemMIMEType>image/jpeg</itemMIMEType>
        </item>
        <item>
            <itemId>DS1</itemId>
            <itemLabel>Thorny's Coliseum thumbnail jpg
image</itemLabel>
            <itemURL>http://localhost:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DS1</itemURL>
            <itemMIMEType>image/jpeg</itemMIMEType>
        </item>
        <item>
            <itemId>DS2</itemId>
            <itemLabel>Thorny's Coliseum medium jpg image</itemLabel>
            <itemURL>http://localhost:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DS2</itemURL>
            <itemMIMEType>image/jpeg</itemMIMEType>
        </item>
        <item>
            <itemId>DC</itemId>
            <itemLabel>DC Record for Coliseum image object</itemLabel>
            <itemURL>http://localhost:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DC</itemURL>
            <itemMIMEType>text/xml</itemMIMEType>
        </item>
        <item>
            <itemId>DS4</itemId>
            <itemLabel>Thorny's Coliseum veryhigh jpg image</itemLabel>
            <itemURL>http://localhost:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DS4</itemURL>
```

```
            <itemMIMEType>image/jpeg</itemMIMEType>
        </item>
    </objectItemIndex>

    <!-- Dublin Core record. Also an attachment with id 'DC'. -->
    <dublinCoreRecord pid="demo:5">
        <dc:title>Coliseum in Rome</dc:title>
        <dc:creator>Thornton Staples</dc:creator>
        <dc:subject>Architecture, Roman</dc:subject>
        <dc:description>Image of Coliseum in Rome</dc:description>
        <dc:publisher>University of Virginia Library</dc:publisher>
        <dc:format>image/jpeg</dc:format>
        <dc:identifier>demo:5</dc:identifier>
    </dublinCoreRecord>
</fedoraItem>
```

Create, search and view pages were also created for a Fedora community in U-P2P. Since Fedora does not have a web-based interface other than through Web Services and the HTTP access methods, creating the stylesheets for displaying resource was much simpler than creating the stylesheets to emulate DSpace.

6.4.5 U-P2P as a Fedora Proxy

A proxy implementation for Fedora was created to allow a peer to provide access to Fedora data. Since Fedora has no web application interface a proxy serves as a useful gateway to a Fedora repository. To implement the proxy, the Repository and DownloadService interfaces were implemented to provide a read-only search and retrieval of objects in a Fedora repository. By only implementing the search methods and not the store or remove methods, the Fedora repository remains completely unchanged and is integrated with U-P2P.

The following sections detail the implementation of the Repository and DownloadService modules for the Fedora proxy. As mentioned in 5.2, the proxy is enabled by changing providers in configuration files before launching U-P2P and once started, the peer remains dedicated to acting as a proxy for the database.

6.4.5.1 Implementing Search in the Fedora Proxy

Searching in Fedora is performed by submitting the search terms to an URL on the Fedora server (http://host:port/fedora/search). The query terms are added to the URL and may query individual supported fields (i.e. Dublin Core attributes) or they can match any field. The maximum number of results to return is also added to the query along with the desired metadata for the returned results. For our purpose only the object PID and title is returned in a search. After submitting the search to the Fedora server, the response is received in XML format. The following is an example of the response to a query:

```
<result>
  <resultList>
    <objectFields>
        <pid>demo:5</pid>
        <title>My Title</title>
    </objectFields>
    <objectFields>
        <pid>demo:3</pid>
        <title>Another Item</title>
    </objectFields>
    ...
  </resultList>
</result>
```

The resulting list of objectFields are parsed and converted into U-P2P search responses and made available on the results page to which the user is directed. The same procedure is followed if a search is either submitted by another peer or if a user is searching locally on the proxy peer. The next step, should a user choose to retrieve one of the search results is to download the resource from Fedora via the proxy peer.

6.4.5.2 Implementing the Fedora DownloadService

The proxy implementation of the DownloadService for Fedora is complicated by the way in which Fedora makes resources accessible. When downloading a resource from Fedora three separate URLs are used: one for the Object Profile, another for the Item Index and

another for retrieving attachments. Normally attachments would be retrieved after the metadata or resource XML, but Fedora stores the Dublin Core record as an attachment that must be retrieved separately.

The following steps are followed to retrieve an item from Fedora (that is the resource XML without attachments):

1. Retrieve the Object Profile for the given PID (which is the objects U-P2P resource ID) using an URL similar to the above search facility. An example Object Profile follows:

```
<objectProfile pid="demo:5">
        <objLabel>Image of Coliseum in Rome</objLabel>
        <objContentModel>UVA_STD_IMAGE</objContentModel>
        <objCreateDate>2004-11-02T13:01:38</objCreateDate>
        <objLastModDate>2004-11-02T13:01:39</objLastModDate>
        <objType>Fedora Data Object</objType>
    </objectProfile>
```

   The Object Profile is appended to a skeleton XML object that will be populated and returned to the user.

2. Retrieve the Item Index from the Fedora server for the same PID. The Item Index lists the attachments associated with the object and is appended to the resource XML to be returned to the user. An example might be as follows:

```
<objectItemIndex pid="demo:5">
        <item>
            <itemId>DS3</itemId>
            <itemLabel>Thorny's Coliseum high jpg image</itemLabel>
            <itemURL>http://10.0.0.0:9080/fedora/get/demo:5/fedora-
system:3/getItem?itemID=DS3</itemURL>
            <itemMIMEType>image/jpeg</itemMIMEType>
        </item>
</objectItemIndex>
```

3. Retrieve the Dublin Core record associated with the item if it is available. The Dublin Core record has the item ID of 'DC' and is retrieved using the same URL

used for retrieving attachments. Once the DC record in retrieved (in XML format), it is parsed and appended to the record being returned to the user.

4. After retrieving all three parts, the record is scanned for attachment URLs (particularly in the Item Index) which are replaced with URLs that point to the proxy peer. This allows the Fedora database to remain anonymous behind the proxy and not receive any requests directly from the peer-to-peer network.

Having completed the above steps, a complete XML resource is returned to the user and the user may subsequently request attachments associated with the item. Attachments are routed through the DownloadService using a simple pipe that takes binary bytes from the attachment on the Fedora server and returns them to the user.

With the Repository and DownloadService interfaces implemented, a Fedora proxy can be set up to bring any Fedora repository into the U-P2P network. No further effort is required to change the data on the Fedora server and only the proxy peers needs read access to the database HTTP interface.

6.5 *Bridging DSpace and Fedora*

DSpace and Fedora communities were created and a Fedora proxy was implemented as the first steps toward integrating two digital repositories. The remaining step was to create a bridge between the two communities and provide mappings between the DSpace and Fedora communities.

The following is the bridge between the DSpace and Fedora communities:

```
<bridge
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
```

```
  <title>DSpace to Fedora bridge</title>
  <description>Relates items in the DSpace community to items in the
Fedora community.</description>
  <comments>The mapping uses a simple translation of XPaths.</comments>
  <bridgeMapping>
    <source>
      <community>d2a9d6f78dcf91828f68a52f78260e05</community>
      <resource>134d8f8ecd57acb35206b4cd13e38622</resource>
    </source>
    <relation>owl:sameAs</relation>
    <target>
      <community>d2a9d6f78dcf91828f68a52f78260e05</community>
      <resource>da1058314b7d8890fc7df7f879a0a7db</resource>
    </target>
  </bridgeMapping>
  <transformList>
    <transform>
      <file>
file://community/BridgeCommunity/DSpaceFedoraXPathMappings.xml</file>
      <type>XPathMap</type>
      <description>XPath mapping for schema elements.</description>
    </transform>
    <transform>

<file>file://community/BridgeCommunity/DSpaceFedoraMapping.owl</file>
      <type>OWL</type>
      <description>OWL mapping from the DSpace community to the Fedora
community.</description>
    </transform>
  </transformList>
</bridge>
```

6.5.1 Mapping DSpace to Fedora

The XPath mappings used to translate DSpace queries to Fedora queries are simple equivalence mappings show below:

```
<XPathMappings>
  <mapping source="dspace/item/metadata/dc:title"
    target="/fedoraItem/dublinCoreRecord/dc:title" />
  <mapping source="dspace/item/metadata/dc:creator"
    target="/fedoraItem/dublinCoreRecord/dc:creator" />
  <mapping source="dspace/item/metadata/dc:subject"
    target="/fedoraItem/dublinCoreRecord/dc:subject" />
  <mapping source="dspace/item/metadata/dc:description"
    target="/fedoraItem/dublinCoreRecord/dc:description" />
  <mapping source="dspace/item/metadata/dc:publisher"
    target="/fedoraItem/dublinCoreRecord/dc:publisher" />
  <mapping source="dspace/item/metadata/dc:contributor"
    target="/fedoraItem/dublinCoreRecord/dc:contributor" />
  <mapping source="dspace/item/metadata/dc:date"
    target="/fedoraItem/dublinCoreRecord/dc:date" />
  <mapping source="dspace/item/metadata/dc:type"
```

```
    target="/fedoraItem/dublinCoreRecord/dc:type" />
  <mapping source="dspace/item/metadata/dc:format"
    target="/fedoraItem/dublinCoreRecord/dc:format" />
  <mapping source="dspace/item/metadata/dc:identifier"
    target="/fedoraItem/dublinCoreRecord/dc:identifier" />
  <mapping source="dspace/item/metadata/dc:source"
    target="/fedoraItem/dublinCoreRecord/dc:source" />
  <mapping source="dspace/item/metadata/dc:language"
    target="/fedoraItem/dublinCoreRecord/dc:language" />
  <mapping source="dspace/item/metadata/dc:relation"
    target="/fedoraItem/dublinCoreRecord/dc:relation" />
  <mapping source="dspace/item/metadata/dc:coverage"
    target="/fedoraItem/dublinCoreRecord/dc:coverage" />
  <mapping source="dspace/item/metadata/dc:rights"
    target="/fedoraItem/dublinCoreRecord/dc:rights" />
</XPathMappings>
```

These mappings are used as described in 5.3.3 to translate a query from DSpace to Fedora. When a query is translated a mapping is required for each XPath that is to be mapped to the target schema. Since Fedora only supports fifteen Dublin Core attributes (the core set with no qualifiers), any additional attributes used in DSpace are not translated and are omitted from the resulting query. This form of query translation is a simplified view of potentially very complex mappings between schema elements. For example, a more complex mapping might want to perform queries over the input data or manipulate the input using algorithms to clean and sort the data.

Additionally mappings for use in translating instance data were not implemented for the case study. Therefore a user can execute their query in the community that is a target of a bridge (e.g. a DSpace query executed in a Fedora community), but they can only view and download the results in the target community. For data integration this effectively points the user toward new sources of data without involving any complex automated searches for bridges or exploration of bridge relationships. This kind of automation and semantic-level mappings are left for future work.

6.5.2 Bridges in the U-P2P Interface

After creating the bridge and uploading it to the Bridge Community, it can be viewed using the View activity in U-P2P. To make finding bridges easier in U-P2, a 'Related Resources' link is now displayed next to the activities of View, Search and Create. By clicking 'Related Resources' a page is displayed with all the bridges that are related to the last displayed resource. This does not trigger a search for new bridges but only displays any current related bridges that are already shared by the user in the Bridge Community. To find new bridges the user can search in the Bridge Community and download new bridges as needed. If the user is not a member of a community that is a bridge end point, the end point cannot be displayed. For example, if a user had the DSpace to Fedora bridge but was not a member of the Fedora community, click on the bridge target would result in a message saying the community is not found. Users can search for the community in the Root Community and download the missing community to avoid this problem. A future version of U-P2P may add the automatic retrieval of missing communities to help the user make the most of bridges.

A link is also displayed after a user has performed a search in a community with a bridge (e.g. the DSpace community). On the search results page the user can click a link to run the same search on the Fedora community or any other community with a bridge and XPath query translation mappings. This triggers the translation of the search query into the Fedora community schema using the XPathQueryTranslation class as mentioned above. After automatic translation, and assuming the user is a member of the Fedora community, the search is dispatched to the Fedora community and search results are displayed with Fedora resources. The search results from Fedora are not combined with

DSpace as yet, but future work on U-P2P integration features may allow a mixed display of results. For now the DSpace to Fedora Bridge with its simple XPath mappings guides the user toward new sources of data but does not collate the results or perform higher-level integrated searches.

6.6 Conclusion

The case study presented in this chapter demonstrates the ability of the improved U-P2P framework to integrate data on a limited basis. DSpace and Fedora are two digital repositories with similar functionality but few provisions (outside of metadata harvesting) for data integration. By creating the communities in U-P2P and writing a bridge to connect the two communities, a query posed in DSpace can be re-written to work in Fedora. Furthermore, a concrete deployment scenario was used for testing the framework and to show its flexibility in deployment.

The mapping language used to map metadata between DSpace and Fedora is limited to simple substitutions and does not have the expressive power of logic statements or more complex mapping languages. This does not mitigate the usefulness of U-P2P as more complex translation modules can be inserted into the framework to take advantage of the latest advances in mapping languages. In addition, the bridge created for the case study does not provide data translation for transforming results back into the original schema used for the search query. This could be added to U-P2P in a future version using XQuery or XSL Transformations.

Chapter 7 Conclusion

The concept of user-contributed mappings to integrate differing data schemas was proposed and implemented within the U-P2P framework. To integrate data, U-P2P had to be modified with several enhancements that allow it to share data across a decentralized network, act as a proxy to existing data sources and create bridges between communities. A fully-distributed network adapter was created to increase the user capacity of the network and to support the use of peers as proxies to external databases. U-P2P was modified to act as a proxy for existing databases. This required extracting the database functionality and modularizing it so other data providers could be used in place of the standard local XML database. By acting as a proxy to external databases peers can add large amounts of existing data to the network to be integrated using the bridge features of U-P2P. A Bridge Community and bridge format was created for U-P2P with source and target communities, a high-level semantic relationship statement and attachments for mappings used in data and query translation. The bridges between communities were integrated into U-P2P with appropriate links placed in the user-interface. The links allow a user to explore related resources and perform a search from one community in another related community through query translation.

A case study was used to show how U-P2P could be used to bridge two digital repositories. The DSpace and Fedora repositories were emulated as communities in U-P2P with schema created for each community and made to be compatible with their native database formats. Create, search and viewing stylesheets were also written for each

community. A proxy was written for Fedora allowing an external Fedora database to join the peer-to-peer network with minimal effort for integration.

The DSpace and Fedora digital repositories used the same metadata but in a different structure and with differing levels of detail. This allowed a one-to-one mapping between compatible metadata to be created and allowed simple XPath translation to be used between two communities in U-P2P.

7.1 Future Work

While the basic infrastructure is implemented in U-P2P for data integration, additional work needs to be done in automating the process of discovering and using bridges. When searching or browsing through resources a user should be able to traverse multiple communities using bridges that translate queries or instance data without intervention from the user. So far, a bridge in U-P2P contains a semantic relationship statement such as `owl:sameAs`. This relationship needs to be integrated with OWL mappings attached to the bridge that describe the ontology of the domain of the integrated data. For example, an OWL document could describe the relationship between each Dublin Core element in the DSpace community to a standardized ontology for libraries and repositories. Software agents operating over U-P2P could then use the semantic relation in the bridge and the attached OWL mappings to reason over the community's data or even build new bridges to other communities.

The peer-to-peer approach to data integration also leaves some unsolved problems that are not specific to U-P2P. Namely, bridges or mappings between schemas are no more robust than local-to-global mappings used in federated or mediated database systems.

Thus a mapping between peers can be superseded just as easily as a mapping to a global schema if the distributed schema changes. However, while a federated global schema can both break and be broken by local schema, a broken mapping in a peer-to-peer system can quickly be replaced by a new mapping contributed by a user. Furthermore, a mapping between two schemas in a peer-to-peer system is not 'broken' but only becomes unusable when a schema referred to in the bridge cannot be found on the network. Due to the redundancy of the peer network, a popular schema will likely continue to be used after a newly revised schema has been published.

While new mappings can be quickly introduced in a peer-to-peer integration system, this leads to another issue not addressed in this thesis: the trustworthiness of data mappings. This thesis did not tackle the challenge of weather to trust a mapping that is found on the network, but future work must consider how to set up networks of trusted information and how to certify or rate bridges (and resource alike) in a distributed system. Working within the U-P2P framework, a Trust Community might be established to share trust information on users or resources. This is contingent on U-P2P establishing secure identities and additional security mechanisms that are left for future work.

References

[Abi98] Abiteboul, S., Duschka, O., "Complexity of answering queries using materialized views", Proc. of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 254-263, 1998, ACM Press

[Art03] Arthorne, N., Esfandiari, B., Mukherjee, A., "U-P2P: A Peer-to-Peer Framework for Universal Resource Sharing and Discovery", Proc. of the USENIX 2003 Annual Technical Conference, FREENIX Track, June 9-14, pp. 29-38, 2003

[Bec04] Beckett, D. (editor), "RDF/XML Syntax Specification (Revised)", W3C Recommendation, 10 February 2004. Retrieved April 26, 2005, from http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[Ben00] Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., Melchiori, M., Vincini, M., "Information Integration: The MOMIS Project Demonstration", Proc. of 26th Intl. Conf. on Very Large Data Bases, September 10-14, 2000, pp. 611-614, Morgan Kaufmann

[Ber01] Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001

[Ber02] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I., "Data mangement for peer-to-peer computing: A vision", Proc. of the 5th Intl. Workshop on the Web and Databases (WebDB'02), 2002

[Bra04] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation, 04

February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-xml-20040204

[Bra99] Bray, T., Hollander, D., Layman, A. (editors), "Namespaces in XML", W3C Recommendation, 14 January 1999. Retrieved on April 26, 2005 from http://www.w3.org/TR/1999/REC-xml-names-19990114

[Bri04] Brickley, D., Guha, R. V., "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Recommendation, 10 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[Cal04] Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M., Rosati, R., "Semantic Data Integration in P2P Systems", Proc. of the Intl Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003), Lecture Notes in Computer Science, Volume 2944, Mar 2004, pp. 77-90, Springer-Verlag

[Chr01] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., "Web Services Description Language (WSDL) 1.1", W3C Note, 15 March 2003. Retrieved on April 26, 2005 from http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[Cla99] Clark, J., DeRose, S., (ed.) "XML Path Language", Version 1.0, W3C Recommendation, 16 November 1999. Retrieved on April 26, 2005 from http://www.w3.org/TR/1999/REC-xpath-19991116

[Cla04] Clark, K. G. (ed.), "RDF Data Access Use Cases and Requirements", W3C Working Draft, 12 October 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/WD-rdf-dawg-uc-20041012/

[Cru02] Cruz, I. F., Rajendran., Sunna, W., Wiegand, N., "Handling Semantic Heterogeneities Using Declarative Agreements", Proc. of the 10th ACM intl. symposium on Advances in geographic information systems (GIS '02), pp. 168-174, 2002

[Cru03] Cruz, I. F., Rajendran., A., "Exploring a New Approach to the Alignment of Ontologies", Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Proc. of the Second Intl. Semantic Web Conference (ISWC03), Sanibel Island, Florida, 20 October 2003

[DCT05] DCMI Usage Board, "DCMI Metadata Terms", Dublin Core Metadata Initiative, 10 January 2005. Retrieved on April 26, 2005 from http://dublincore.org/documents/2005/01/10/dcmi-terms/

[DCL04] DCMI-Libraries Working Group, "Dublin Core Library Application Profile (DC-Lib)", Dublin Core Metadata Initiative, 09 October 2004. Retrieved on April 26, 2005 from http://dublincore.org/documents/2004/09/10/library-application-profile/

[Doa03] Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., Halevy, A., "Learning to match ontologies on the Semantic Web", The VLDB Journal - The Intl. Journal on Very Large Data Bases, Volume 12, Issue 4, pp. 303-319, 2003, Springer-Verlag

[Dsp05] DSpace digital repository, MIT and Hewlett-Packard (HP). Retrieved on April 26, 2005 from http://dspace.org/introduction/index.html

[Dus97] Duschka, O., Genesereth, M., "Answering recursive queries using views", Proc. of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, ACM Press

[Fal04] Fallside, D., C., Walmsley, P., "XML Schema Part 0: Primer", Second Edition, W3C Recommendation, 28 October 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/

[Fri99] Friedman, M., Levy, A., Millstein, T., "Navigational Plans for Data Integration", Proc. of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), July 1998, Orlando, Florida, United States, pp. 67-73, 1999

[Fra03] Franconi, E., Kuper, G., Lopatenko, A., Serafini, L., "A robust logical and computational characterization of peer-to-peer database systems", Proc. of the VLDB Intl. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03), 2003

[Fra04] Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I., "The coDB Robust Peer-to-Peer Database System", Proc. of the 2nd Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGrid'04), 2004

[Gar97] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J., "The TSIMMIS Approach to Mediation: Data Models and Languages", Journal of Intelligent Information Systems, Volume 8, Issue 2, pp. 117-132, March 1997

[Gnu05] The Gnutella Developer Forum, Clip2, "The Annotated Gnutella Protocol Specification v0.4". Retrieved on April 26, 2005 from http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[Goh99] Goh, C. H., Bressan, S., Madnick, S., Siegel, M., "Context interchange: new features and formalisms for the intelligent integration of information", ACM Transactions

on Information Systems (TOIS), Volume 17 , Issue 3, pp. 270-293, July 1999, ACM Press

[Hal01] Halevy, A., "Answering queries using views: A survey", The Intl. Journal on Very Large Data Bases, Volume 10, Number 4, December 2001, pp. 270-294, 2001, Springer-Verlag

[Hal03a] Halevy, A., Ives, Z., Tatarinov, I., Mork, P., "Piazza: Data management infrastructure for semantic web applications", Proc. of the Intl. World-wide-web Conference, Budapest, Hungary, 20–24 May 2003, pp 556–567

[Hal03b] Halevy, A., Ives, Z. G., Suciu, D., Tatarinov, I., "Schema Mediation in Peer Data Management Systems", Proc. of the 19th Intl. Conference on Data Engineering (ICDE'03), pp. 505-516, March 2003

[HCS04] House of Commons Science and Technology Committee Tenth Report, "Scientific Publications: Free for all?", HC 399-I, United Kingdom, 20 July 2004. Retrieved on April 26, 2005 from http://www.publications.parliament.uk/pa/cm200304/cmselect/cmsctech/399/399.pdf

[Hef04] Heflin, J., "OWL Web Ontology Language Use Cases and Requirements", W3C Recommendation, 14 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-webont-req-20040210/

[JXT05] The JXTA Project, originally of Sun Microsystems, Inc.. Retrieved on April 26, 2005 from http://www.jxta.org

[Kaz05] Kazaa file sharing client (uses the FastTrack P2P stack), Sharman Networks Ltd.. Retrieved on April 26, 2005 from http://www.kazaa.com

[Kly04] Klyne, G., Carroll, J. J., "Resource Description Framework (RDF): Concepts and Abstract Syntax", W3C Recommendation, 10 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[Lag02] Lagoze, C., Van de Sompel, H. (editors), "Open Archives Initiative Protocol for Metadata Harvesting, Protocol Version 2.0, 14 June 2002. Retrieved on April 26, 2005 from http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

[Len02] Lenzerini, M., "Data Integration: A Theoretical Perspective", Proc. of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, Madison, Wisconsin, USA, pp. 233-246, 2002, ACM Press

[Lim05] LimeWire P2P File Sharing Program (Gnutella 0.6 protocol), LimeWire LLC. Retrieved on April 26, 2005 from http://www.limewire.com

[Lit85] Litwin W., "An overview of the multidatabase system MRDSM", Proc. of the 1985 ACM annual conference on The Range of Computing: mid-80's Perspective, pp. 524-533, 1985

[Lit90] Litwin W., Mark, L., Roussopoulos, N., "Interoperability of multiple autonomous databases", ACM Computing Surveys (CSUR), Volume 22, Issue 3, pp. 267-293, September 1990

[Luc05] Jakarta Lucene, an open-source full-text search engine written in Java, Apache Software Foundation. Retrieved on April 26, 2005 from http://jakarta.apache.org/lucene

[Lyn03] Lynch, C. A., "Institutional Repositories: Essential Infrastructure for Scholarship in the Digital Age", Association of Research Libraries (ARL) Bimonthly Report 226, February 2003. Retrieved on April 26, 2005 from http://www.arl.org/newsltr/226/ir.html

[Man04] Manola, F., Miller, E. (editors), "RDF Primer", W3C Recommendation, 10 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[McG04] McGuinness, D. L., van Hermelen, F., "OWL Web Ontology Language Overview", W3C Recommendation, 10 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-owl-features-20040210/

[MET05] Metadata Encoding and Transmission Standard (METS), Library of Congress Network Development and MARC Standards Office, 26 April 2005. Retrieved on April 26, 2005 from http://www.loc.gov/standards/mets/

[MOD05] Metadata Object Description Schema (MODS), Version 3.0, Library of Congress Network Development and MARC Standards Office, 26 April 2005. Retrieved on April 26, 2005 from http://www.loc.gov/standards/mods/

[Muk02] Mukherjee, A., Esfandiari, B., Arthorne, N., "U-P2P: a peer-to-peer system for description and discovery of resource-sharing communities", Proc. of the 22nd Intl. Conference on Distributed Computing Systems Workshops, pp. 701-705, 2002

[Mus05] EchoMine Muse, a Jabber and Gnutella API written and implemented by Chris Chen. Retrieved on April 26, 2005 from http://open.echomine.org/cowiki/

[Nap05] The Napster file-sharing service has since been replaced by a digital music store. Retrieved on April 26, 2005 from http://www.napster.com

[Noy00] Noy, N. F., Musen, M., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment", Proc. of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, Texas, United States, 2000, AAAI Press

[OAI05] Open Archives Initiative (OAI), Mission Statement. Retrieved on April 26, 2005 from http://www.openarchives.org/organization/index.html

[Ost05] OAIster, University of Michigan Digital Library Production Service. Retrieved on April 26, 2005 from http://oaister.umdl.umich.edu/o/oaister/

[Pat04] Patel-Schneider, P. F., Hayes, P., Horrocks, I., "OWL Web Ontology Language Semantic and Abstract Syntax", W3C Recommendation, 10 February 2004. Retrieved on April 26, 2005 from http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[Pin01] Pinto, H. S., Martins, J. P., "A methodology for ontology integration", Proc. of the intl. conference on Knowledge Capture, Victoria, British Columbia, Canada, 2001, pp. 131-138, ACM Press

[Pow03] Powell, A., Johnston, P., "Guidelines for Implementing Dublin Core in XML", Recommendation, Dublin Core Metadata Initiative, 2 April 2003. Retrieved on April 26, 2005 from http://www.dublincore.org/documents/2003/04/02/dc-xml-guidelines/

[Rot96] Roth, M. T., Arya, M., Haas, L., Carey, M., Cody, W., Fagin, R., Schwarz, P., Thomas, J., Wimmers, E., "The Garlic Project", Proc. of the 1996 ACM SIGMOD intl. conference on Management of data, pp. 557, 1996, ACM Press

[She99] Sheth, A., Interoperating Geographic Information Systems, chapter "Changing Focus on Interoperability in Information Systems: from System, Syntax, Structure to Semantics", Goodchild M.F., Egenhofer, M.J., Fegeas, R., Koffman, C.A. (eds.). Kluwer Academic Publishers, Boston, pp. 5-29, 1999

[She90] Sheth, A., Larson, J., "Federated database systems for managing distributed, heterogeneous, and autonomous databases", ACM Computing Surveys (CSUR), Vol. 22, No. 3, pp. 183-236, September 1990

[Sta03] Staples, T., Wayland, R., Payette, S., "The Fedora Project: An Open-source Digital Object Repository System," D-Lib Magazine, April 2003. Retrieved on April 26, 2005 from http://www.dlib.org/dlib/april03/staples/04staples.html

[Tha01] Thadani, S., "Meta Information Searches on the Gnutella Network", LimeWire LLC. Retrieved on April 26, 2005 from http://rfc-gnutella.sourceforge.net/developer/testing/metadata.htm

[Web05] W3C Web Services Activity (see individual specifications). Retrieved on April 26, 2005 from http://www.w3.org/2002/ws/

[Wie92] Wiederhold, G., "Mediators in the architecture of future information systems", IEEE Computer Vol. 25, No. 3, pp. 38-49, March 1992

Appendix A: Bridge XML Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="bridge">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string" />
      <xsd:element name="description" minOccurs="0" type="xsd:string"
/>
      <xsd:element name="comments" minOccurs="0" type="xsd:string" />
      <xsd:element name="bridgeMapping">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="source">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="community" type="xsd:string" />
                  <xsd:element name="resource" type="xsd:string" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="relation" type="xsd:QName" />
            <xsd:element name="target">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="community" type="xsd:string" />
                  <xsd:element name="resource" type="xsd:string" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="transformList" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="transform" minOccurs="0"
maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="file" type="xsd:anyURI" />
                  <xsd:element name="type" type="xsd:string" />
                  <xsd:element name="description" type="xsd:string" />
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```