U-P2P: A Peer-to-Peer Framework for Universal Resource Sharing and Discovery

Neal Arthorne, Babak Esfandiari, Aloke Mukherjee
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
narthorn@connectmail.carleton.ca
babak@sce.carleton.ca
alokem@cisco.com

Abstract

We present U-P2P, an open source framework for developing, deploying and discovering file-sharing communities. We address the problem of search in peer-to-peer file sharing by allowing the end user to add metadata to shared documents. Each file-sharing community allows the sharing of a particular structured document type. Communities are themselves modeled as structured documents, thus enabling their sharing and discovery just like any other document. The creator of a particular community specifies, among other properties, the document type that it shares and the deployment model. U-P2P's extensible architecture allows developers to create new properties or extend existing ones. For example, developers can provide new deployment models or custom privacy and authentication features. U-P2P makes use of other open source projects such as Jakarta Tomcat and eXist, an XML database system.

1 Introduction

The current success of peer-to-peer (P2P) file sharing applications has highlighted the benefits of resource distribution and redundancy. However, to truly exploit such advantages, a few roadblocks remain to be cleared. In particular, search and discovery of resources is still quite difficult. Most known approaches rely on simple schemes such as a search for the resource name or type. File sharing communities are most efficient when the name of the file carries most if not all of the needed information. As a result, most file swapping communities are restricted to swapping music and video files. Even when the exchange of non-music files is possible, the difficulty of finding such files has been a sufficient deterrent. Lack of metadata is the main problem.

Another roadblock to more general applications of P2P has been the difficulty of creating communities for specific purposes. This arises partially from the difficulty of defining custom metadata about different types of files. Another important problem is the current fractured state of peer-to-peer communities. Again, the lack of metadata about communities makes it difficult to know what there is to look for in the first place.

We propose a peer-to-peer framework called Universal Peer-to-Peer (U-P2P) that simplifies the sharing of custom metadata formats as well as the easy creation, configuration and discovery of peer-to-peer communities. In U-P2P, each community is described in part by the metadata of the files it exchanges. The format of a community's metadata is specified using the XML Schema language, allowing new communities centered on that file type to be created in any text editor. U-P2P allows these custom resources to be created and shared as in traditional file-sharing services.

By logical extension, the description of the community itself (the metadata format of its files, the protocol used for search, etc) is encapsulated in an XML file. In U-P2P, creating, sharing or discovering a community follows the same principles as creating, sharing or discovering a file within that community. As a result, file-swapping communities such as Napster or Kazaa can be seen as *instances* of U-P2P devoted to sharing a few specific file types and utilizing a given P2P protocol.

2 Related Work

Our focus is on the discovery problem in peer-to-peer systems. In Napster [1], the only files that could be shared on the network were MP3 audio files. Search was based on filenames and relied on users encoding the artist and title of each song in the MP3's title. Although metadata such as encoding rate could be used to sort the results, there was no way to search on these metadata or define other parameters. On a Gnutella [2] network, any type of file can be shared: however there is no explicit metadata handling. Search strings are passed around without processing between peers and their interpretation is left to the peer. Each peer must implement its own search algorithm using the search string

as input. Most Gnutella implementations, like Napster, simply return filenames that contain the search string. Gnutella does permit the design of overlay protocols to encode and decode metadata from search strings. Some Gnutella developers have proposed using this approach for richer metadata searches. [3, 4]. Schemas are defined for common file types: for example an audio file might be defined to have properties such as artist, title, bit rate, album, etc. The schema defines a structured format for searching MP3 metadata that is sent as a search string to other Gnutella nodes. Responding clients use the query to search local files annotated using the schema and returns the results using the same structured format. In practice though, all members of a given community must be able to speak a common language in order to communicate.

Other P2P systems such as FastTrack [5], Opencola [6] or Bitzi [7] propose variations on that idea, but they are still limited to a number of predefined schemas. The latter two have the capability to extract metadata information from a file given the file format, but this is only possible for certain formats.

Clearly, there is a need for *shared ontologies* if we want to allow search for any type of file. In the Internet world, the XML Schema format[8] is the current choice to represent ontologies, replacing Document Type Definition (DTD) [9], the schema language defined in the original XML specification. XML Schema supports the creation of custom and complex data types for XML tags, which is essential to describe aggregated or composite resources. For richer semantic descriptions, for example to allow software agents to perform search instead of humans, there can be a need to describe relationships between resources. RDF [10] and more generally the Semantic Web [11] effort address that need. The Edutella project [12] is a technology for distributed learning that uses a P2P network for sharing RDF-formatted metadata. However in Edutella metadata is the resource, not the means to describe one.

A P2P system using a semantic layering approach is not without its drawbacks. First and foremost is getting users of the system to supply metadata for their resources. The addition of metadata requires user-friendly tools for authoring RDF and schemas and a simplified approach for users who are not familiar with XML languages.

What we propose in U-P2P starts, as in Edutella, with the sharing and discovery of metadata described this using XML Schema. Once metadata is discovered it is used to instantiate a particular resource, which is in turn shared. The next section gives a high-level description of the principles behind U-P2P as well as its design.

3 Background: XML Schema

U-P2P relies heavily on XML Schema and the following sections include some examples of XML Schema and a brief description of the structure of XML Schema documents.

XML Schema is a Recommendation [8] by the World Wide Consortium for a schema language for XML. It constitutes a set of markup tags themselves written in XML, that are used to describe both the structure of an XML document and the datatypes of individual elements and attributes. XML Schema uses the XML Namespace Recommendation that allows authors to prefix their own tags with a specific namespace. It also allows importing types from other schemas both in the current namespace and from other namespaces.

XML Schema can be used to validate XML documents using a schema processor. XML documents themselves can link to their schema using a schemaLocation tag. However this is only a suggested method for specifying a schema for the document and they may use other types of schemas such as DTDs or not require validation at all.

In XML Schema, there are four basic units that can be used to describe parts of an XML document: elements, attributes, simpleTypes and complexTypes.

Elements - Elements are the tags used in XML to label a piece of data. They have a datatype associated with them that is either defined at the same time as the element (anonymously) or is a complexType or simpleType. Elements can contain subelements, attributes and can also be part of an XML Namespace. The structure of an element can be controlled in XML Schema using sequences, choices and the same restrictions used to derive simple and complex types. For example an author can specify that an element called 'name' should consist of an exact sequence of the two elements 'firstname' and 'lastname'.

Attributes - Attributes are properties that belong to a parent element. They can only use simpleTypes and they cannot contain any elements or attributes. Attributes can be optional or mandatory within an element.

Simple Types - These types are either built-in data types specified in the XML Schema Recommendation or Simple Types derived from the built-in or existing types (XML Schema defines some derived types such as date and time). The Recommendation includes many useful data types such as string and decimal, that can easily be manipulated into more specialized Simple Types.

Complex Types - These types are a sequence or choice between a set of elements and/or attributes. They are reusable (when defined on a global level) and can be used in any part of the current schema. Once a Complex Type is defined an element can use it by specifying the

type attribute equal to the name of the Complex Type. An example of a Complex Type follows:

```
<xsd:complexType name="CanadaAddress">
    <xsd:sequence>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="postalCode">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:pattern="\w\d\w\s\d\w\d"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="province" type="xsd:string"/>
      <xsd:element name="country" type="xsd:string"
      fixed="Canada"/>
    </xsd:sequence>
</xsd:complexType>
```

An example conforming to this schema:

```
<CanadaAddress>
  <street>1125 Colonel By Drive</street>
  <city>Ottawa</city>
  <postalCode>K1S 5B6</postalCode>
  <province>Ontario</province>
  <country>Canada</country>
</CanadaAddress>
```

Having defined the above example, the type can then be used in an element anywhere in the schema or even referenced from other schema.

XML Schema is powerful enough to describe any complex XML structures and the support for data types means the contents of shared documents are validated using the lexical representation of the data. When sharing large volumes of documents, it is useful to have full validation at the entry of the system as this will prevent the spread of unstructured data that can't be properly searched and indexed.

4 U-P2P Concepts

U-P2P provides four fundamental services: *search, create, browse local* and *view*. Each of these services are provided in the context of a community. We can imagine the existence of a "stamps" community that trades pictures and descriptions of stamps from around the world. On entering the stamp community the U-P2P Search function offers fields to search for stamps of a given year, and/or from a given country. Similarly, the Create function prompts you to upload a picture of the stamp, Browse Local shows the stamp objects that you have already downloaded and View displays a picture as well as the attributes for one of your downloaded stamps.

In traditional P2P applications this functionality would require downloading a client that knows about the format of a stamp object and contains customized search, create and view screens for such an object. In U-P2P, we use the power of metadata to simplify this task. Consider the following XML schema describing a stamp object:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://w3.org/2001/XMLSchema">
<xsd:element name="stamps">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="desc" type="xsd:string"/>
      <xsd:element name="picture" type="xsd:anyURI"/>
      <xsd:element name="country" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

The schema above describes the expected elements of a stamp object and their data types. It is possible to generate a form from this specification using XML Stylesheet Language Transformations (XSLT) [13]. Here is an excerpt from a stylesheet that generates a *search* form from the above XML schema:

```
<xsl:template
  name="SchemaTemplate"
  match="*[local-name()='schema']">
<h3>Search for a Resource</h3>
<p>Enter keywords in any of the fields
below to perform a search.</p>
<form action="search" method="post">
<table border="1" cellpadding="5" cellspacing="0">
  <tr><th>Property</th><th>Value</th></tr>
  <xsl:for-each select=
    "descendant::*[local-name()='element'
                and count(./child::*) = 0]">
    <xsl:call-template name="ElementTemplate"/>
  </xsl:for-each>
</table>
<p><input type="hidden" name="up2p:community">
  <xsl:attribute name="value">
  <xsl:value-of select="$communityName"/>
  </xsl:attribute>
</input>
<input type="submit" value="Search"/></p>
</form>
</xsl:template>
```

Similarly, stylesheets can be produced that render Create forms or display a stamp object. With nothing more than a few XML documents (a schema and stylesheets for creating, searching and displaying), it is possible to define a whole new P2P file-sharing application! In fact, U-P2P provides default stylesheets for handling display and forms for resources made up of common types, making the transformation processes transparent to the novice user. Figure 1 shows the relationships between the U-P2P functions, and how they are accessed through XSL transformations.

So how can a stamp community be found or shared in the first place? The idea in U-P2P is to see a file-sharing community as just another type of resource. This is analogous to the idea of a class in object-oriented programming which specifies the structure of objects. In pure object-oriented languages such as Smalltalk, a class is merely another type of object whose structure is specified by a metaclass. Traversing the analogy in the opposite direction, a specific U-P2P community can be seen as a class instantiated by a more general metaclass: **a**

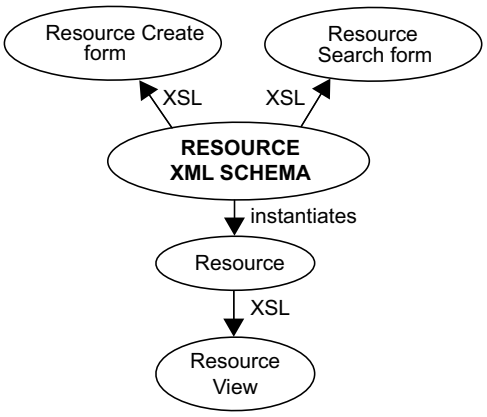

Figure 1: Generation of resource-specific displays

Community-sharing community (in short: a community community) **shares Community objects**:

- an *object* is an instance of its *class*, which is an instance of its *metaclass*
- *mp3* belongs to *mp3 community*, which belongs to *community community*

In U-P2P the problem of discovering the existence of a community is thus reduced to the problem of finding an object. This provides a standard way to discover the existence of resource-sharing communities.

To facilitate this, U-P2P comes packaged with one ”bootstrap” schema that can be used to search for and more importantly create communities:

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="community">
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="displayLocation"
             minOccurs="0" type="xsd:anyURI"/>
            <xsd:element name="searchLocation"
             minOccurs="0" type="xsd:anyURI"/>
            <xsd:element name="createLocation"
             minOccurs="0" type="xsd:anyURI"/>
            <xsd:element name="schemaLocation"
             type="xsd:anyURI"/>
            <xsd:element name="name"
             type="xsd:string"/>
            <xsd:element name="category"
             minOccurs="0" type="xsd:string"/>
            <xsd:element name="keywords"
             minOccurs="0" type="xsd:string"/>
            <xsd:element name="description"
             minOccurs="0" type="xsd:string"/>
            <xsd:element name="protocol"
             minOccurs="0" type="protocolType"/>
        </xsd:all>
        <xsd:attribute name="title" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

<xsd:simpleType name="protocolType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value=""/>
      <xsd:enumeration value="Generic Central Server"/>
      <xsd:enumeration value="Gnutella"/>
      <xsd:enumeration value="JXTA"/>
    </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

As can be seen, a community can have many attributes: keywords, deployment protocol, security... This means that a community can be created by choosing specific values for such attributes (e.g. `keywords: stamps, Canadian; deployment: Napster-style`). The search for a community is made in similar fashion, by filling out a similar form. As of now however, we only provide one type of deployment protocol, ”Napster-style”, in which the peer that creates the community also acts as a broker. Other deployment models, such as ”Gnutella-style” (no broker required) and ”centralized repository” (a client-server option with no local copies of files) are currently being developed. This means that searching for a document could either be completely decentralized, or that on the other extreme full persistence of files could be assured by a central storage of files. Deployment decisions are thus left entirely up to the creator of the given community. Also, we have not yet explored various security or privacy schemes. Such possibilities are discussed later in this paper, in the section on design. The modular aspect of our design should hopefully allow the open source development community to provide support for many more of these attributes.

The schema combined with default stylesheets as described above allow U-P2P to become an engine for creating and searching for communities which trade all sorts of different types of files. A stamp collector using U-P2P for the first time will go to the community search form and type ”stamp” into the keyword field. Upon finding a community of collectors, he or she might download the community including the community’s schema and stylesheets. U-P2P then offers the option of entering the community. Once in the community the collector can perform all the actions one would expect of a file-sharing application devoted specifically to sharing stamps.

Figure 2 shows snapshots of a stamp view, a stamp search form and finally the root community view, the highest level of abstraction in U-P2P.

5 U-P2P Architecture and Design

Like other file-sharing services, U-P2P consists of a client and a file server running on a user’s computer. The client part of U-P2P is implemented using Java Server Pages (JSPs)[15] running on a local web server. The prototype uses Jakarta Tomcat [16], but any server capable of serving JSPs may be used. The user connects to the U-P2P network by pointing their browser at the address of the local web server, typically `localhost:8080`.

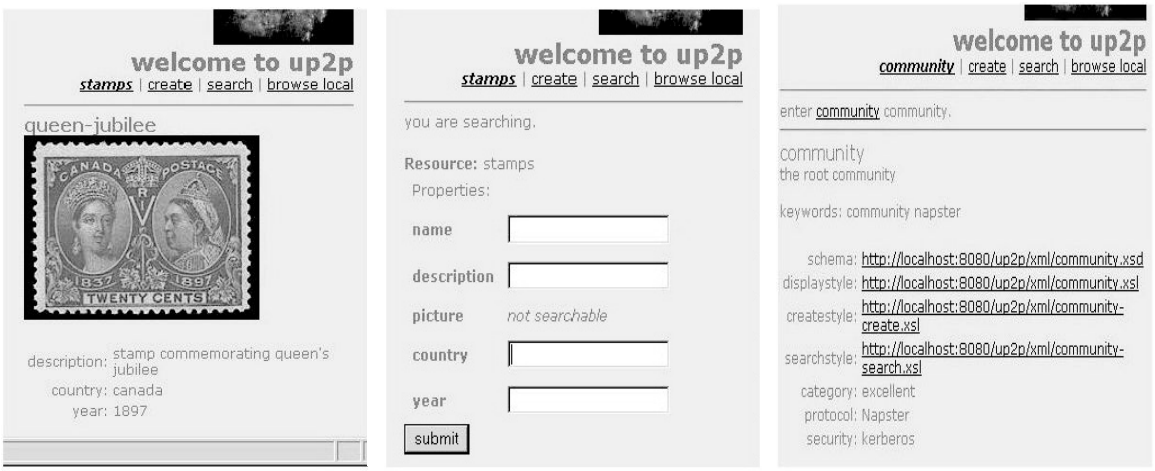

Figure 2: The "stamps" community

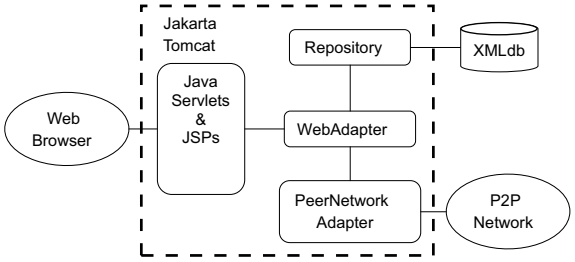

Figure 3: The U-P2P Architecture

The web server acts as GUI, and dispatches *search* and *create* requests to the other peers. In the current prototype, U-P2P follows the Napster model. This means that there is also a central server that acts as a database for information about all shared objects in the system. As with Napster, the information about the location of files is stored centrally but file transfers are conducted between peers. We are considering other possible peer configurations, such as the Gnutella distributed model and a hybrid one like FastTrack.

5.1 Architectural Model

U-P2P consists of three major components shown in 3: the WebAdapter, PeerNetworkAdapter, and the Repository. These components form the core of U-P2P and provide all the services needed to share and discover resources on a Peer-to-Peer network.

WebAdapter - Glues the components together and provides a single point of access for the user interface. This component currently supports a browser-based GUI, but could be replaced for alternate GUIs.

Repository - Stores all shared XML resources in a persistent XML database (using the XML:DB Database API [17]) and provides local search capabilities to the WebAdapter and the PeerNetworkAdapter. This allows for distributed P2P topologies: each node must be able to execute searches against its own set of shared resources.

PeerNetworkAdapter - Provides an interface to the underlying Peer-to-Peer network and is responsible for servicing search requests, publishing resources and downloading resources from the network.

The above components are modeled as Java interfaces, with their implementations as DefaultWebAdapter, DefaultRepository and GenericPeerAdapter respectively. Additional classes include:

FileMapper - Maps resource IDs to real files on the local file system. When a file is 'uploaded' it is assigned an ID and mapped without modifying the file. The FileMapper is persistent in case of shutdown of the U-P2P client and file mappings are restored on startup.

FileMapEntry - Holds a reference to a resource file and all its attachments, pulled from the resource in the upload process. Attachment names must be unique within a resource. Resource IDs are generated from the content of the XML file using an MD5 hashing function. When hashing, a special ResourceProcessor is used that omits any attachment links within the XML. This allows the hash to stay consistent when the links are changed by another peer upon download of the resource.

BasePeerNetworkAdapter - A skeleton class that holds a reference to a DefaultRepository and implements the accessor for the repository. The GenericPeerAdapter is the generic P2P implementation included with U-P2P, which follows a Napster-type model. Any developer wishing to provide an alternative peer-to-peer deployment, such as a fully distributed one, or one that would plug into an existing network, would have to provide a different adapter.

DatabaseAdapter - Performs the dirty details needed

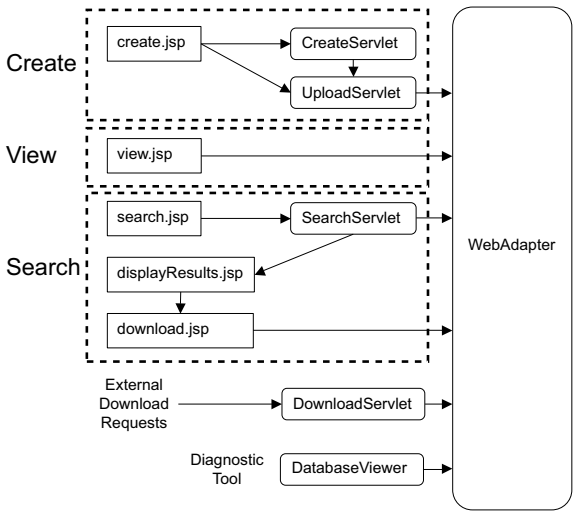

Figure 5: U-P2P Servlets

to get an XML database up and running and to configure the port that it runs on. The current implementation uses eXist 0.8, an open source XML database [18]. If a switch were made to a different XMLdb implementation, this class would be sub-classed or replaced.

The class diagram in figure 4 illustrates the relationships between these classes.

The web-based user interface requires that dynamic pages be served up to the user for such activities as *Search*, *Create* and *View*. The general flow of events that occur in one of these activities involves the user accessing a JSP, the JSP submitting a form to a Servlet and the Servlet talking to the WebAdapter and then returning through a JSP.

Figure 5 shows the pages and Servlets involved in each activity as well as the two extra Servlets needed for diagnostic reasons and for servicing download requests.

5.2 Security in U-P2P

In peer-to-peer systems, data integrity, authentication and authorization are concerns that are shared with other network communication systems. In U-P2P we are concerned with a layering of meta-data that is used on top of an existing network that may or may not be secure. For this reason, the bulk of security measures are left up to the network adapter used to communicate with the underlying network. Each peer network has its own requirements for security. Thus it would not be suitable for U-P2P to impose a minimum level of security for all networks using the U-P2P layering: this would restrict the ability to join public, non-secure networks such as Gnutella or Freenet. Instead, it is up to the founder of a community to decide which network adapter to use: the level of security provided by the adapter will then be used in all network communication.

It is reasonable to assume that a set of standard adapters could be made available alongside a secured version of each adapter. The currently available centralized peer-to-peer adapter could for example, communicate through Secure Sockets Layer SSL [19] channels and wrap all XML resources with an XML Signature [20]. This would ensure that the content of the resources have not been tampered with while in transit or when shared by another user. The current Tomcat 4 platform used by U-P2P has full provisions for SSL communication, but the current release of U-P2P is not secured. U-P2P also uses the Java Servlet standard that provides role-based security suitable for deployment and integration with existing infrastructure.

It should be noted that the current implementation of U-P2P uses MD5 sums to generate a unique ID when a resource is first uploaded to the network. This ID is not intended for security purposes, but as a simple means to check if multiple users are sharing the same resource. In a future release of U-P2P, the core of U-P2P could make use of the MD5 sum and XML Signatures for integrity and authentication of shared resources, with the security of communications remaining in the network adapter.

6 Case Studies

Two separate case studies have been developed to study the usefulness and the performance aspects of U-P2P. The first case study attempts to solve the known problem of searching for design patterns. However, as the number of design patterns that are properly documented and formatted is not sufficient to observe the behavior of U-P2P under high load, another series of tests were run using the Genome database.

6.1 Pattern Repository

The Carleton Pattern Repository [21] was started in 1999. It served as a repository for software design patterns and provided extensive search capabilities over a small list of patterns. The patterns were represented in XML using a DTD designed especially for the repository project [22]. The original pattern repository project included plans to expand the repository to a distributed model.

The distributed model proposed was for each author group to have a repository server with a fixed list of the other servers in the network. The servers would form a highly distributed mesh and send out their searches to all other servers. This model was not implemented and evidently, no one else has pursued the idea.

Using the DTD as a basis, we have developed an XML Schema for representing design patterns [23]. This is used as the basis of a file-sharing community for design patterns. In addition to the schema a custom stylesheet is

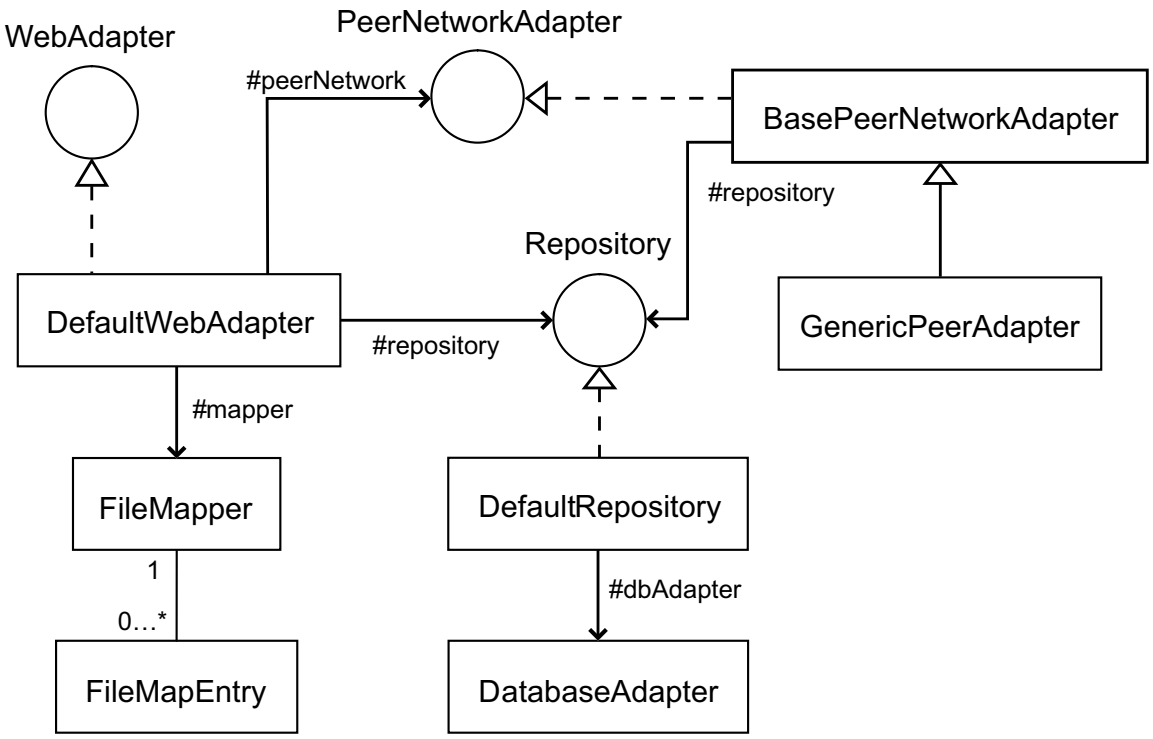

Figure 4: U-P2P core classes

required to render this complex object since the default stylesheet is tailored to simpler formats. Another design problem is deciding which parts of the design pattern should be indexed. For simplicity, our current implementation of U-P2P uploads the entire XML pattern file to make it fully searchable.

To our knowledge, prior to our work there has been no way to share design patterns in a peer-to-peer fashion that incorporates meta-data search. When fully implemented this U-P2P based system will expand the benefits of peer-to-peer file-sharing to this area. Such a system would allow computer scientists and students to publish a rich collection of patterns into an underlying peer-to-peer network, search them using rich queries and replicate popular patterns to increase their accessibility. The community-discovery aspect could also be used to access sub-communities devoted to different classes of design patterns or based on different underlying networks.

6.2 Drosophila Genome Project

A performance evaluation was made using a client and server running on a single computer. The performance characteristics for creating objects and searching were measured. Running on the same node removed the effect of network latency and emphasizes the delays incurred by the software and the XML database. There were two major challenges to measuring U-P2P's performance characteristics. The first challenge was having multiple communities, each with a large collection of objects. The second was automating the process of publishing and searching for objects.

Creating a community and populating it with files manually was too time-consuming so we sought out archives containing large numbers of XML objects. One example of such a repository can be found at the Berkeley Drosophila Genome Project (BDGP) [24] This project maintains a database of annotations of different parts of the Drosophila (fruitfly) genome. The project makes these annotations available in different forms including as a large XML file. This file is split such that each annotation was stored in an individual file. Another pre-requisite for U-P2P communities is a valid XML schema. As with many existing XML repositories, the BDGP repository is described using the older DTD technology. This has been transformed into XML Schema with the help of the dtd2xs [25] package.

Although the BDGP collection was not intended for a peer-to-peer environment, it does present one possible peer-to-peer application. The BDGP is part of FlyBase, a distributed project involving scientists at many different institutions. Each annotation contains a collection of information that can be enriched as information is gathered. With U-P2P each institution (or even each scientist) could maintain their own local database of annotations. By searching on a specific gene's name these multiple annotations could be discovered, viewed and

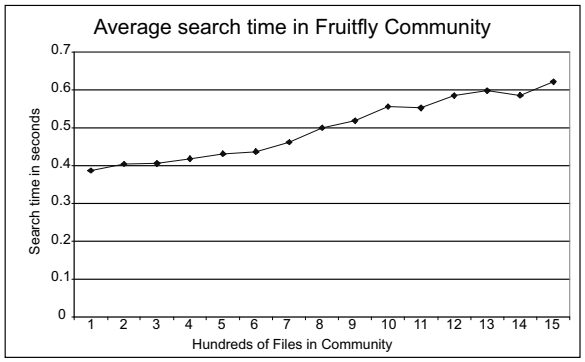

Figure 6: Evolution of search time based on community size

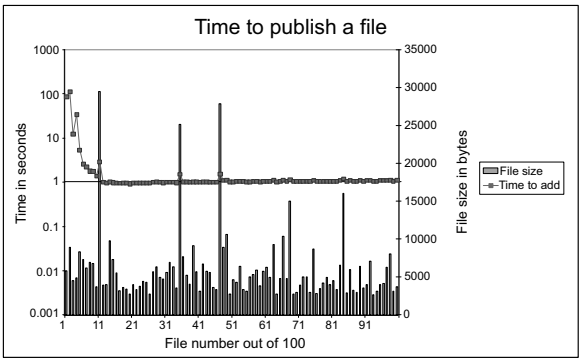

Figure 7: Time to publish based on community size

dynamically reconciled, rather than requiring curation of each annotation at a central repository.

Manually publishing and searching for large numbers of files was not practical, so these operations were automated. An XMLRPC interface was designed for the publish function. A command-line XMLRPC client was written to take advantage of this capability allowing the rapid addition of multiple objects. Search used a different approach, relying on the URL to encode an XPath search expression for a servlet designed to handle XPath formatted queries. These two command-line tools can then be used to script more complex experiments.

The time to publish and search was measured as repository size increases. One hundred files were published to the Fruitfly Community. Measurements are shown in Figures 6 and 7. 6 shows that publishing time increases in a linear fashion with number of files created. A set of fifty search queries was executed after one hundred files were added to the community. The same queries were repeated every one hundred files, up to fifteen hundred files. Finally each set of fifty execution times for various search queries was averaged and plotted versus the number of files. Figure 7 illustrates that the initial publishing of objects is costly: two orders of magnitude slower than the steady state time to publish. An explanation for this is the overhead required to create a new collection, a new object and whatever other internal data structures are used by the XML database. Eventually this time settles down to a steady state. On a Pentium 4 system this value was 1s.

Another interesting set of experiments was performed to determine whether publish and search times are affected by the presence of other communities. The first experiment 8 measures the time to add the first 10 files to a community as the number of existing communities is increased from none to three. Each community was created using 100 or so objects: the Fruitfly community was then created and the time to add the first 10 files recorded.

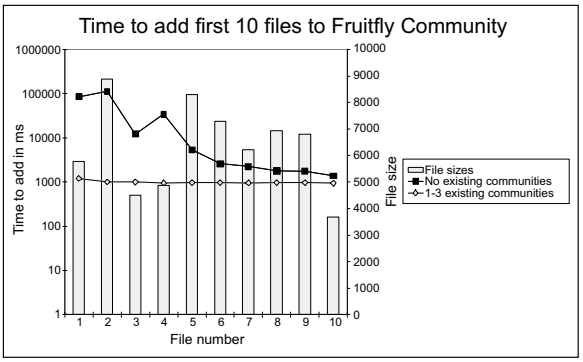

Figure 8: Time to publish when increasing the number of communities

Adding a file when there are no published objects incurs a performance penalty but these quickly converge to the 1s figure mentioned previously. Adding the first object to a community in the presence of other communities does not incur this penalty. Publishing times for objects in the new community are comparable to the time it would take to add more objects to one of the existing communities. This indicates that it is the total number of objects in the system and not the number of objects in a community which determines how long it will take to publish a new object.

In Figure 9, four searches were performed in the Fruitfly community and the time for results to return recorded. The same four searches were performed within the community in the presence of between one and three other communities (each with about 100 files). The first of the searches took about twice as long without the presence of other communities than with, but the subsequent three searches took about the same time regardless of the number of communities. This indicates that search times converge more quickly than publish times and that the existence of other communities does not affect search times.

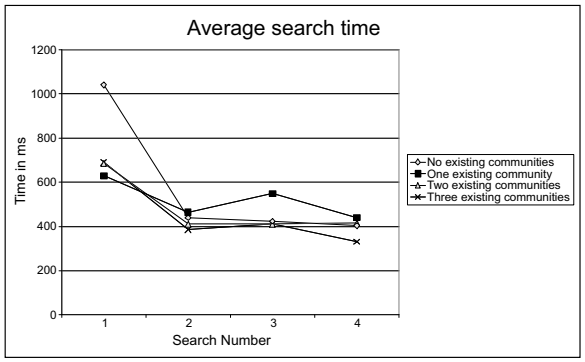

Figure 9: Search time when increasing the number of communities

6.3 Further Comments

A permanent U-P2P server has been set up (`http://24.102.27.174:3300/up2p`) and the Fruitfly gene annotations have been published there. In addition three other collections of XML documents have been published:

- DBLP [26] - references to Computer Science related papers, theses and proceedings
- EDGAR [27] - filings with the US Securities and Exchange Commission (SEC)
- Gene Ontology [28] - database of gene-related concepts

These are all pure XML communities: objects do not contain embedded objects. A few hundred objects have been published in each community. These can be searched by accessing the client running at the permanent location or via a U-P2P peer installed elsewhere. In addition, Chemistry Markup Language (CML) researchers also attempted to create a community for sharing molecules using this server.

In general these preliminary performance results indicate that base performance is a general problem: publish times of 1s on a Pentium 4 are too slow. However, the quick convergence of publish and search times, their slow linear growth as the number of files increase and practical experience indicate that a U-P2P node is scalable and can accommodate a few thousand files before becoming too slow to be useful. One approach to remedying this problem would be to upgrade to the latest version of XML:DB which promises performance enhancements. Another option would be to use the file system for storing documents although this makes rich search more difficult. One concrete improvement is the ability to designate other nodes to be central servers for new communities. This offloads requests as well as storage space allowing the network as a whole to serve many more objects.

7 Conclusion and Future Work

U-P2P is a peer-to-peer framework that allows a user to describe, share and discover communities just like any other resource. Once a community is found, its schema and associated stylesheets are downloaded and are used to perform search and publishing of resources specific to that community. Communities play a generative role similar to metaclasses in object-oriented languages. Possible applications of U-P2P include sharing resources such as resumes, knowledge management in a corporate setting, or distributed repositories for design patterns and software components.

A major direction for future work is in demonstrating the protocol independence of U-P2P. By developing PeerNetworkAdapters to interface to existing networks such as Freenet or Gnutella, U-P2P could become a meta-data layer that would provide an enhanced community-based search capability.

U-P2P communities may also be extended beyond the schema format to include definition of parameters such as protocol, security and authentication. These various parameters define a design space for peer-to-peer systems. Instead of envisioning a single system that is all things to all people, a more practical approach is to consider which methods are most appropriate to the application. The system should allow important features such as security or authentication to be decoupled and mixed and matched.

Despite some database performance issues that we believe can be overcome, as it stands U-P2P can be easily used for distributed sharing of XML documents, while exploiting structured metadata to optimize search.

Availability

U-P2P is an open source application licensed under the GPL, and makes use of other open source products such as Jakarta Tomcat [16] and eXist [18]. Complete source code and documentation as well as guides and presentations are accessible at `http://u-p2p.sourceforge.net`.

Acknowledgments

This work is supported by the Natural Sciences and Engineering Research Council of Canada and the Foundry Program of Carleton University. Valuable feedback on U-P2P has been received from Peter Murray-Rust and his team at the University of Cambridge, UK. The design pattern case study would not have been possible without the help of Darrell Ferguson and Dwight Deugo of the Carleton School of Computer Science.

References

[1] Napster, http://www.napster.com Accessed on 07 April 2003 06:00 UTC.

[2] Gnutella, http://gnutella.wego.com Accessed on 07 April 2003 06:00 UTC.

[3] Thadani, Sumeet, *Meta Information Searches on the Gnutella Network*, http://www.limewire.com/index.jsp/metainfo_searches Accessed on 07 April 2003 06:00 UTC.

[4] Thadani, Sumeet, *Meta Data searches on the Gnutella Network (addendum)*, http://www.limewire.com/developer/MetaProposal2.htm Accessed on 07 April 2003 06:00 UTC.

[5] FastTrack, now owned by Sharman Networks and found in Kazaa Media Desktop http://www.kazaa.com Accessed on 07 April 2003 06:00 UTC.

[6] OpenCola project, http://www.opencola.com Accessed on 07 April 2003 06:00 UTC.

[7] Bitzi, http://www.bitzi.com Accessed on 07 April 2003 06:00 UTC.

[8] XML Schema Part 0: Primer, W3C Recommendation, 2 May 2001 http://www.w3.org/TR/xmlschema-0/ Accessed on 07 April 2003 06:00 UTC.

[9] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000, http://www.w3.org/TR/REC-xml Accessed on 07 April 2003 06:00 UTC.

[10] Resource Description Framework, W3C http://www.w3.org/RDF/ Accessed on 07 April 2003 06:00 UTC.

[11] Tim Berners-Lee, James Hendler and Ora Lassila, *The Semantic Web*, Scientific American, May 2001.

[12] Nejdl, Wolfgang et al, *EDUTELLA: A P2P Networking Infrastructure Based on RDF*, http://edutella.jxta.org/reports/edutella-whitepaper.pdf Accessed on 07 April 2003 06:00 UTC.

[13] Extensible Stylesheet Language Transformations (XSLT) Version 1.0, W3C Recommendation, 16 November 1999, James Clark (Editor), http://www.w3.org/TR/xslt Accessed on 07 April 2003 06:00 UTC.

[14] Gong Li, *JXTA: A Network Programming Environment*, IEEE Internet Computing Vol 5. No. 3. May/June 2001

[15] JavaServer Pages, Sun Microsystems, http://java.sun.com/products/jsp/ Accessed on 07 April 2003 06:00 UTC.

[16] Jakarta Tomcat, Apache Software Foundation, http://jakarta.apache.org/tomcat Accessed on 07 April 2003 06:00 UTC.

[17] XML:DB API, Working Draft, 20 September 2001, Kimbro Staken (Editor), http://www.xmldb.org/xapi/xapi-draft.html Accessed on 07 April 2003 06:00 UTC.

[18] eXist 0.8, http://exist.sourceforge.net Accessed on 07 April 2003 06:00 UTC.

[19] The SSL Protocol Version 3.0, Internet-Draft, 18 November 1996, http://wp.netscape.com/eng/ssl3/draft302.txt Accessed on 07 April 2003 06:00 UTC.

[20] XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002, http://www.w3.org/TR/xmldsig-core/ Accessed on 07 April 2003 06:00 UTC.

[21] Dwight Deugo, Darrell Ferguson, Carleton Pattern Repository, http://muffin.nexus.carleton.ca/~darrell/repo/ Accessed in July 2002.

[22] Dwight Deugo, D. Ferguson, *XML Specification for Design Patterns*, Proceedings of the Second International Conference on Internet Computing (IC'2001): 407-412.

[23] Neal Arthorne, *An XML Schema for Design Patterns*, http://chat.carleton.ca/~narthorn/project/patterns/pattern.xsd Accessed on 07 April 2003 06:00 UTC.

[24] G.M. Rubin, *Around the Genomes: The Drosophila Genome Project*, Genome Research (1996) 6:71-79

[25] DTD2XS, http://puvogel.informatik.med.uni-giessen.de/lumrix/ Accessed on 07 April 2003 06:00 UTC.

[26] DBLP, http://dblp.uni-trier.de/xml/ Accessed on 07 April 2003 06:00 UTC.

[27] EDGAR, http://bulk.resource.org/edgar/xml/ Accessed on 07 April 2003 06:00 UTC.

[28] The Gene Ontology Consortium, *Gene Ontology: tool for the unification of biology* Nature Genetics 25: 25-29.