Using Peer-To-Peer Networks For a Component Repository Search System

Kevin Lam, Babak Esfandiari

Carleton University, Ottawa, Ontario, K1S 5B6

Abstract

Software components are key to rapid "plug and play" application development. Components are typically found in many web-based repositories, which presents a challenge to software developers searching for a component to suit their particular needs. A developer must search numerous repositories looking for candidate components, and then evaluate them to discover a suitable one. Most current web-based repositories do not provide very advanced search facilities, forcing developers to examine possibly dozens of potential components in order to find one suited to their needs. A component repository built as a peer-to-peer network, such as currently popularized by music-sharing systems (Napster, Kazaa, Gnutella, etc.) may offer a solution that provides access to a very large distributed collection of components. To provide domain specific search results, an ontology-based classification and search system should also be developed. Such a network can also incorporate other useful services such as direct user-to-user communication or collaboration, or search integration with development tools. We present a design for a component repository using peer-to-peer networking, and develop prototypes using two peer-to-peer network frameworks, "Universal Peer To Peer" (U-P2P) and Project JXTA. We evaluate the prototype repositories primarily by their ease of use and ease of implementation, particularly from an end-user perspective. Our results illustrate many of the advantages of peer-to-peer networks for the sharing of software components, but may also be extended to sharing of any resources that can be described with metadata. Finally, we suggest that U-P2P may be implemented as an application on top of JXTA, as a tool for the rapid deployment of a variety of resource-sharing applications in a peer-to-peer fashion.

1 Introduction

The move from procedural, low level programming code to object-oriented methods signaled a sort of "industrial revolution" in the software industry. A similar paradigm shift is rapidly emerging through the development and deployment of software components: modular, self-contained and "plug and play" units, which can be composed together to construct new applications [13]. Components allow a developer to maximize code reuse, reduce cost and decrease development time.

It has been estimated that the commercial market for software components was USD $7 billion in 2001, expanding to up to $64 billion in 2002 [8]. Many thousands of existing components are available, written in dozens of implementation languages (such as ActiveX, JavaBeans, etc.). The number of components has become so vast that a common challenge for developers is finding and retrieving the components they wish to use [13, 19].

Typically a developer searches for a component in one or more *component repositories*. These are often found on the Internet, but many enterprises and other organizations may also operate their own repositories [11].

The challenge of finding all suitable components within a repository is compounded by the fact that many dozens of repositories exist. A developer typically needs to search many repositories to ensure that a suitable component is found. (Likewise, component vendors typically submit their components to many repositories in order to maximize exposure.) In addition, the searcher must have some knowledge of the repository search space, i.e. one must search using the same keywords or terms as those indexed in the repository [19]. Finally, once potential matches are found and duplicates are filtered, the developer must evaluate each result to determine if it is suitable for their needs, which itself can be time consuming [7].

1.1 Existing Component Repositories

Many component repositories currently exist. Some may be privately operated for the internal use of organizations (including government bodies). Others are publicly accessible, most of which are commercial collections [19]. Typically a repository is simply a web-based collection of components incorporating a simple keyword search. Some collections also manage other resources, including documentation, tutorials, discussion forums, etc.

Table 1 lists a number of public component repositories available on the World Wide Web, and outlines some of their key characteristics including the number of components in the repository, search facilities, etc.

These web-based component repositories share a number of limitations, as can be seen from the table:

- **repository size**: individual repositories each contain a relatively small number of components, or are of limited scope.
- **limited search options**: most existing libraries do not provide very advanced search capability, often limited to keyword matching in text descriptions.
- **no interoperability**: there is generally no standard format for the component database or the metadata, making it difficult to merge or transfer databases.
- **duplication across repositories**: users searching for components must search across many collections. Likewise, vendors must submit their components to multiple repositories to maximize the component's exposure.
- **version control**: vendors must track which repositories contain which versions of their software and update every repository whenever a component is updated.
- **expressions of suitability (ranking)**: different repositories have their own methods (if any) for characterizing the popularity or suitability of components for particular applications. Even so, the user typically must still install and evaluate every component that potentially matches their requirements.
- **user interface**: there is no standard interface for searching, browsing, or submitting components. Sites often require membership for advanced use, requiring multiple accounts and passwords.

Much research has been done to develop better component repositories (e.g. [15], [8], [7], particularly [16]). Researchers commonly place efficiency, effectiveness, and a useful user interface as high priorities (e.g. [16]). Repository projects of varying nature exist, but not all provide a clean user interface, and some require the use of proprietary tools (e.g. [15], [7], [18].) Charters [8] acknowledges that their system requires a better way to visualize non-functional attributes of components such as security and reliability. Some of the repository-development methods proposed rely on the availability of source code and documentation ([8], [16]), which in many cases are proprietary and not available to public systems.

1.2 The ideal solution?

A perfect solution would perhaps be to have one central repository, infinitely scalable and

Description/URL	Interest	Size	Search Fields	Metadata	Standards?	Summary
www.active-x.com, "Your Active-X Component Library Since 1996" - collection of controls for sale by numerous vendors	Commercial	2000	keyword search in category, can select to search company name, summary, description	title, version, description, summary, URL, requirements, price, license type, category, component type, screenshot	no	Aimed at software developers, submissions by individuals or companies are not tested or reviewed. Users may rate each entry. Primarily Microsoft technologies (Active X, ASP, VBScript)
Jfind.com, "Your Ultimate Java Software Directory" - collection of Java applets, classes, Beans, and other tools	Commercial	2263	keyword search	title, short and long descriptions, URL, company, license types, score (rating)	no	Aims to be the largest Java directory available
directory.google.com/Top/Computers/Programming/Component_Frameworks/ Google Component Frameworks directory	Search Engine (individually submitted or discovered)	500	none (but can be searched by Google)	Category, description, URL	Google database	Taken in context, this is just a small portion of Google's Web Directory.
AX2NET.com ActiveX/COM directory	Commercial	100-200	none	category, description, summary, URL, logo image, version, platform, price, license type	no	Listings include icon indicators. Site also includes resources, message forum
VisualBuilder.com, "Community for Multi-Skilled Developers" - many platforms, languages, resource types	Commercial	500-1000	keywords, with filters	category, description, URL, version, size, price, author, ratings	no	Multiple "channels" for different languages and component types, includes much more than just software (tutorials, forums, articles), also links to commercial venture ComponentSource.com
www.open-components.org/ Community/ Repository/ index.jsp Java Component Repository	University open project	535	keywords	category, description, author, URL, date, version	no	Open source project by Chinese University of Hong Kong; database records contain 1..* flags
www.cs.colorado.edu/yunwen/ codebroker/ CodeBroker - an active component repository system	PhD thesis project	n/a (framework only)	automatically generated keyword queries	description, signature	Java (free-text files)	Project aiming to equip an IDE with ability to automatically search a component repository for existing code based on perceived needs of program being written

Table 1: Features of several component repository projects available on the Internet

expandable, to contain every available software component. Every new component would be submitted to the repository by its vendor, and every user would look there for components, eliminating duplication. The repository would contain a rich classification system to enable efficient browsing and targeted searching for components, with a ranking system to help users determine the quality and usefulness of particular components. It would be built upon open standards to facilitate upgrading and enhancement. An efficient search engine would quickly return precise search results to reduce time spent on component evaluation (e.g. a user would only need to evaluate 5 "very likely" matches rather than 20 "good" matches).

It is unrealistic to expect that such a central repository could exist, due not only to physical-access issues such as bandwidth, server storage space, and server scalability, but also economic ones (proprietary "in house" components, commercial implications, competition, etc.)

2 Peer to Peer Networks

In recent years there has been an explosion of research and public awareness into the area of peer-to-peer networks. This has been fuelled largely by the success of music and file-sharing applications such as Napster, Gnutella, and Kazaa. Ethical issues aside, these applications have become extremely successful because, simply, they are effective. Music-sharing networks routinely share very large numbers of files between thousands of users. One study of the Gnutella network identified a total of 1,239,487 unique peers (not all were online simultaneously). Around 75% of peers shared 100 or fewer files; 7% shared 1000 or more. [10]

Generally, peer-to-peer file sharing networks offer a number of advantages over the traditional client-server (multiple clients to one server) approach:

- **Distributed search and large repository size**: A peer can query many systems (other peers) simultaneously rather than sending a single query to a monolithic server. If each peer is well populated with files, this becomes equivalent to searching multiple standalone file repositories in parallel. In addition, any file present on one peer will be accessible via many peers. The total number of files accessible can greatly exceed what would be possible with a traditional standalone server [10]. Ci [9] suggests that distributed search across multiple repositories provides better performance than a single central repository.
- **Dynamic membership and scalability**: Peer-to-peer file sharing networks typically generate dynamically in an ad-hoc manner. Thus the network is easily expandable and, subject to the constraints of the network architecture, scalable. It is also generally resilient to network or peer failures, as requests are simply routed to other peers. Moving a set of files from one peer server to another is completely transparent to the network.
- **Reduced (or no) reliance on central servers**: Files are kept with peers, which are distributed across the entire network. There is no need for one server to consume a large amount of bandwidth or file storage space. The degree of decentralization varies between implementations [10]; some may require a central query server.
- **Redundancy**: There will often be some overlap between content shared by various peers. While this can still be a disadvantage (similar to the duplication found among multiple standalone repositories), it is of less consequence in a peer-to-peer environment since the level of redundancy is immediately visible in one set of search results. File redundancy can be advantageous, allowing users to choose which of several peers to retrieve a file from, based on current network conditions (highest bandwidth, lowest network latency, etc.) The degree of redundancy in a peer-to-peer network can even be a measure of file popularity.
- **Popularity**: A number of peer-to-peer networks have already well-established user bases. Some are specifically optimized for sharing digital music (e.g. Napster), others are capable of sharing any files.

- **Direct peer communication**: With no distinction between client and server in a peer-to-peer network, any peer may contribute resources and communicate directly with other peers (users). Users are generally not limited to file sharing - many peer-to-peer implementations allow direct user-to-user messaging or collaboration. This is a powerful concept not available in traditional client-server systems.

Peer-to-peer networks are not without their limitations. Network infrastructures vary between implementations, and some (e.g. Gnutella) are more vulnerable to network problems (or deliberate attacks) than others [10]. Network latency and bandwidth can also become an issue since ad-hoc "user" peers may not have the same computing resources or bandwidth available as a full-time file server. Some networks partially overcome this during file transfers by seeking two or more peers hosting an identical file and splitting the transfer between them.

3 Using Peer To Peer Networks For Component Repositories

We will examine the use of peer-to-peer networks to build a distributed component repository, effectively providing a single point of entry to a myriad of component collections. While still not an approximation of the "ideal world" central repository, a peer-to-peer repository system can address many of the limitations associated with existing repositories and also have the potential to provide other useful and powerful services such as direct user-to-user collaboration.

3.1 Network Topology

A peer-to-peer file-sharing network could be used as the framework for a component repository. Similar to music-sharing networks, it would allow large collections of files (and possibly other resources) to be aggregated through the peers that own them. While typical peers would be end users who connect only for a short time (Napster and Gnutella had median connection times of approximately 60 minutes [10]), "server-like" peers analogous to standalone repositories may remain permanently connected to the network.

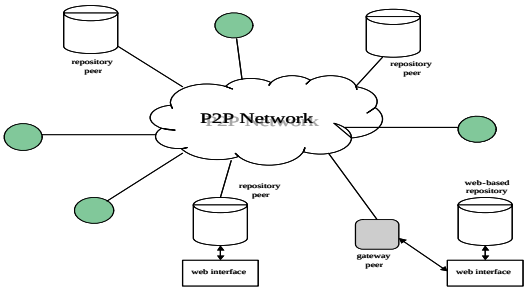

Figure 1: A peer-to-peer component repository topology including client (search) peers, repository peers (which may also provide a web interface), and gateway peers to interface with web-based repositories not directly connected to the network

The success of a peer-to-peer repository over traditional approaches depends largely on the number of components that would be made available through it. Ideally the network would completely contain the domain of all existing web-based component repositories. A web-based repository, for example, might also be a peer in the network, allowing access to its files from either HTTP or P2P.

An existing web-based repository could join a peer-to-peer network either by running additional software on the server, turning it into a native client of the network, or through a web-based gateway, allowing network access by proxy. A "gateway" peer, used to interface the repository to the peer-to-peer network, would receive search queries from the peer network, reformulate them to match the particulars of the web repository it interfaces with, and submit the search to the web server. The results would be retrieved, translated into the network's native messaging format, and passed back to the querying peer.

Figure 1 illustrates a topology for such a network including the various types of peers that could participate.

The possibility exists for the network to include other types of peers, including peers in-

tegrated into development tools. For example, Ye and Fischler describe in [18] a system called CodeBroker, in which the programming environment automatically searches a component repository based on the code currently being entered by the user. Such a system could plug into a peer-based network, allowing access to the repository through these development tools. Users of such a tool could collaborate directly with each other, sharing components, solutions to common bugs, or other insights - even if such users were working on completely different projects.

3.2 Categorization and Search

The other key factor in establishing a large-scale component repository is the ability to categorize and quickly query for components. The search facility should improve upon that which is offered by typical web-based repositories, many of which offer only a basic keyword search in text descriptions.

Music sharing networks (e.g. Napster, Kazaa) offer keyword searches based attributes such as a song title or artist name. Clearly, software components may contain many such defining attributes, for example:

- component name
- component version
- component description
- hierarchical category
- implementation language (ActiveX, Java, C++, etc.)
- platform and system requirements
- pricing and license information
- vendor contact information

An ontology-based classification system may be used to fully exploit the metadata offered by component vendors (including the attributes above). This would allow for detailed search queries that yield higher precision and recall [19].

3.3 Performance Measurements

It should be possible to measure the relative performance and merits of a peer-to-peer component repository against a traditional approach to determine the extent of improvement. Factors to consider include:

- number of unique accessible components
- search speed
- specificity of queries (i.e. ability to precisely narrow down a search)
- robustness to network changes or congestion
- ability to rate components (popularity measure)
- value-added services (peer messaging, etc.)
- ease of use

These metrics are both qualitative and quantitative. Speed and repository size are clearly important, as are the ability to generate specific queries and the ease of which the repository may be accessed.

4 A Prototype Peer To Peer Component Repository

The following sections describe the development of a prototype repository for advertising and sharing software components. Instead of trying to develop yet another proprietary peer-to-peer network from scratch, the goal was to develop the prototype using an existing framework.

We begin with an evaluation of suitable peer-to-peer network software. Most existing peer-to-peer networks are specifically designed for sharing only a few types of files (music, video) and are thus unsuitable for use in building a component repository. Some, like Gnutella, are able to share any file, but are limited in ability to describe files or search for files (typically only looking at filenames). While it would be possible to share components using such a platform, it would not support component-specific

4.3 Component Sharing With U-P2P

The classification above was implemented in U-P2P using its default "generic" peer-to-peer server implementation. Installing the U-P2P software requires only downloading the binary package and executing the included start-up script.

A total of five definition files in XML and HTML need to be created to fully define a software component community. They are listed in table 4.

Creating these HTML/XML files is all that is necessary to completely define the new community. Once created, the U-P2P web interface was used to register the new community with the server. New components can then be defined by filling in the information on the HTML form (figure 2). Components can also be specified by creating an XML advertisement and providing U-P2P with a reference to the file (filling in the HTML form automatically generates such an advertisement). Searching for a component is equally straightforward - the HTML search form is used and results are displayed using the XSLT stylesheet.

Figure 5 shows the XML schema used to define a component advertisement according to the classification data defined previously in table 4.

While no code development is required to define a custom community in U-P2P, the steps required are not trivial. One must define an XML schema for advertising the desired resources, create two HTML forms, and develop an accompanying XSLT file. Finally (optionally) a community definition file is created which is registered to U-P2P. XML tools are available to semi-automate the process. A user experienced with HTML and XML can create and deploy a new community in less than an hour.

The U-P2P component repository prototype was up and running in a very short time. U-P2P's ease of use and GUI interface make it ideal for prototyping and file sharing, however because it is based entirely on a browser interface it lacks some of the services offered by other peer to peer frameworks (like JXTA), such as peer instant messaging. The included generic peer to peer engine is also somewhat slow to respond to queries and resource creation.

While the U-P2P prototype demonstrates a simple repository built on peer-to-peer network technology, the U-P2P system does not eliminate several of the problems associated with standalone repositories. U-P2P by itself relies on a central server for queries and registration of all resources available on a network. This effectively negates many of the benefits of using peer-to-peer networks including distributed search and scalability to large repository sizes. The goal of the U-P2P project is to adapt its interface to run on top of a decentralized system such as Gnutella or JXTA, where these limitations would largely disappear.

4.4 Component Sharing With JXTA

Project JXTA itself is not an application but a framework for developing peer-to-peer applications. JXTA provides protocols, communications services, and other building blocks from which a component file-sharing application could be built.

We evaluate several possibilities using JXTA at its various levels: platform level, service level, and application level. Because the goal is to be able to develop a custom resource-sharing application as quickly and easily as possible, a prototype was only developed where it was reasonably simple to do so.

4.4.1 JXTA Platform

JXTA, at the platform level, provides only a set of protocols, API's, and other building blocks for developers to build upon in order to create a peer-to-peer application. It supports the creation and joining of peer groups and communication between peers.

JXTA's core platform is versatile enough to support the development of a wide range of applications, from voice-over-IP and live music streaming to multiplayer and board games [3]. All of these, however, require that additional code to be developed over top of the JXTA core libraries. JXTA's core platform does not include any kind of executable en-

Category	Project JXTA	U-P2P
Implementation Language	Java	Java
XML-based?	Yes, though XML documents are abstracted away from the user	Yes, end user must write XML documents to use in U-P2P
Application	open platform and development framework for general peer to peer networking between "any" digital devices	file and resource sharing on top of an existing P2P platform
Layers (Scope)	low-level services (transport, search, user management), many application-level projects are under development	user interface (i.e. high level) for defining communities, searching and browsing of resources
User Interface	low-level services accessed through JXTA shell; other user interfaces may be developed on top of JXTA at the application level	browser-based, with customizable GUI for manipulating communities and resources
Equivalent Terminology (in resp. order)	peer groups, rendez-vous, pipe	communities, server, connection

Table 2: Comparison of various characteristics of U-P2P and Project JXTA

Category	Attributes
Identification	id, name, textDescription, author, location
Platform	version, language, developmentTool, requirements
Management	category, relatedComponents, classification
Registration	price, licenseType, vendorAddress, telephone, email, vendorURL, registrationDate

Table 3: A set of classification metadata for components

Filename	Description
components.xml	A definition file for the "components" community (may also be automatically generated using the U-P2P web interface). This file contains references to the four files below, and describes the name and a description of the community.
components.xsd	An XML schema defining the characteristics of a component resource (i.e. its advertisement). This schema describes XML documents which contain the metadata shown in 3
components-create.html	An HTML document displayed when creating a new component resource to be shared in the peer network. This is an HTML form with input fields for each metadata attribute.
components-display.xsl	An XSLT stylesheet used to visualize the component advertisement in the web browser, translating the metadata elements into HTML text, links, image links, etc.
components-search.html	An HTML document displayed when querying for resources in the component community. This is an HTML form with input fields for each metadata attribute.

Table 4: Files created to define a component community in U-P2P

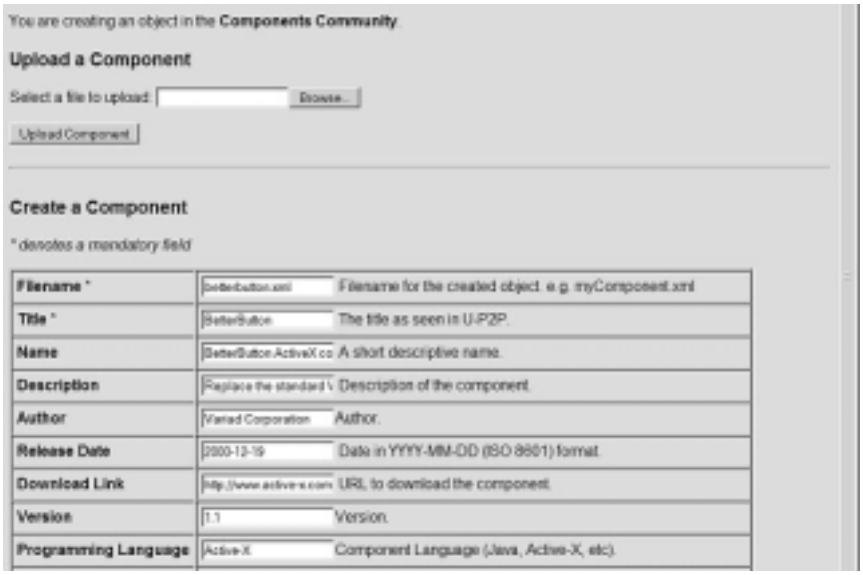

Figure 2: Screenshot of the "Create Component" web form in U-P2P

```
<?xml version="1.0" encoding="UTF-8"?> <xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <xsd:element
name="components">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="releaseDate" type="xsd:date"/>
      <xsd:element name="download" type="xsd:anyURI"/>
      <xsd:element name="version" type="xsd:string"/>
      <xsd:element name="language" type="xsd:string"/>
      <xsd:element name="category" type="xsd:string"/>
      <xsd:element name="componentType" type="xsd:string"/>
      <xsd:element minOccurs="0" name="devTool" type="xsd:string"/>
      <xsd:element minOccurs="0" name="requirements" type="xsd:string"/>
      <xsd:element minOccurs="0" name="relatedComponents" type="xsd:string"/>
      <xsd:element name="price" type="xsd:string"/>
      <xsd:element name="licenseType" type="xsd:string"/>
      <xsd:element minOccurs="0" name="vendorURL" type="xsd:anyURI"/>
      <xsd:element minOccurs="0" name="vendorAddress" type="xsd:string"/>
      <xsd:element minOccurs="0" name="vendorPhone" type="xsd:string"/>
      <xsd:element minOccurs="0" name="vendorEmail" type="xsd:anyURI"/>
      <xsd:element minOccurs="0" name="picture" type="xsd:anyURI"/>
    </xsd:all>
    <xsd:attribute name="title" type="xsd:string"/>
  </xsd:complexType>
</xsd:element> </xsd:schema>
```

Table 5: XML schema for component advertisements

vironment, though the command-line JXTA Shell does provide low-level access to most of the platform functions such as creating and joining a peer group or sending text messages to peers.

To build a repository upon core JXTA services, one would need create a new application from scratch. The application would connect to the JXTA core services to create a peer group devoted to the component repository, and define a peer group advertisement suited to describing resources shared in the group. The user would need to develop the interface (GUI based, or extending the JXTA Shell) as well as the code to manage communication pipes and file transfers between peers. While resources such as peer group and content advertisements are defined in an XML messaging format, the XML code is hidden from the user, encapsulated through Java API method calls.

As many JXTA services and applications already exist which have already implemented some form of file sharing or content management, attempting to build a repository directly on top of the JXTA core would be akin to reinventing the wheel. Several existing services could be adapted or extended for the task of sharing components.

4.4.2 JXTA Content Management System

One of the JXTA services under development is the JXTA Content Management System (CMS). CMS defines a Content Advertisement which can be used to describe resources in an XML format resembling the following:

```
<?xml version="1.0">
    <!doctype jxta:contentAdvertisement>
    <jxta:contentAdvertisement>
    <name>index.html</name>
    <cid>md5:1a8baf...b7a83</cid>
    <type>text/html</type>
    <length>23983</length>
    <description>Web site index</description>
    </jxta:contentAdvertisement>
```

While the content advertisement cannot be expanded to include other attributes, CMS does provide a metadata API allowing for additional keywords or text descriptions. Currently, however, searching for content described using Content Advertisements is still limited to substring searching in the `<name>` and `<description>` fields; searches do not examine the content metadata. The CMS development site [2] suggests that this is an upcoming feature.

Developing an application on top of this service still requires a custom-built GUI and file management system to input, display and download components. Components would be described by a content advertisement, using extended metadata to describe component attributes. The CMS service would provide ability to search for specific content and return matching advertisements to the repository application, which would then display results to the end user and provide a means for component retrieval. However, the current CMS search limitation severely limits the usefulness of CMS as a search service, as searches cannot currently be made based on a component's attribute metadata.

Several existing JXTA applications concentrate on sharing files, built on top of the CMS service. In the following sections we will discuss the most promising ones.

4.4.3 JXTA Search

The JXTA Search service is extremely well suited to the task of sharing components, particularly those currently hosted on web-based repositories. JXTA Search provides a distributed "deep search" facility where search queries of user-customizable complexity can be created using the Query Routing Protocol, and sent to peers either directly on a JXTA network or through HTTP-based queries to web servers. Query results are generated by the peer and sent back in XML format. Providers (peers or web servers) register for particular queryspaces and receive only queries that match them. Most nodes on the network would be provider peers, coordinated by a number of hub peers responsible for resolving and routing of queries and responses. While one hub peer could potentially handle many thousands of requests at once, additional hub peers can be used to provide a distributed and decentralized search service [17].

The JXTA Search comes by default with a web-based client interface which allows network searches to be done via a web browser. The

JXTA Search interface is thus intuitive and easy to use, much like U-P2P. JXTA Search allows application-based queries along with web-based ones, so other applications could be written which interface to the JXTA Search service and send queries to the network directly.

The ability for JXTA Search to communicate with web-based providers is analogous to the concept of "gateway peers" described earlier. A web server needs only to execute a script (written in a language such as PHP, JSP, or ASP) which would parse an XML query document, perform a search based on the search terms provided, assemble an XML document with the results, and post it back to the querying peer. Any web server, including the host server of an existing component repository, could run such a script to allow those components to be searchable via the JXTA Search network.

Adapting JXTA Search to work with components would be straightforward, similar in scope to what was required to develop a component sharing system in U-P2P. All a JXTA Search node or web server would have to provide is an XML registration document in order to register it with a JXTA Search hub. A browser-based console is provided to facilitate registrations. A registration for queries formatted to resemble the component metadata specified in 3 would look like the following:

```
<?xml version='1.0'?>
<register xmlns="http://search.jxta.org"
 xmlns:b=http://lotsofcompos.com/jxtasearch >
  <query-server>
    http://lotsofcomponents.com/jxtasearch.php
  </query-server>
  <query-space uri= http://lotsofcompos.com/jxtasearch >
  <predicate>
    <query>
      <b:name></b:name>
      <b:description></b:description>
      <b:title></b:title>
      <b:author></b:author>
      <b:releaseDate></b:releaseDate>
      <b:download></b:download>
      <b:version></b:version>
      <b:language></b:language>
      <b:category></b:category>
      <b:componentType></b:componentType>
      <b:requirements></b:requirements>
      <b:relatedComponents></b:relatedComponents>
      <b:price></b:price>
      <b:licenseType></b:licenseType>
      <b:vendorAddress></b:vendorAddress>
      <b:picture></b:picture>
      ... (some tags omitted)
    </query>
  </predicate>
  </query-space>
</register>
```

The web-based GUI interface would then be modified to include HTML search forms for each of the above attributes, which would generate an XML search query. Responses to the query would be formatted in a similar manner, each describing all matches found by its respective provider. In the case of a JXTA Search provider peer, support is built-in for generating such responses; in the case of a web-based repository, the responses would be generated by a server-side script. The server would send an HTTP response to the POST request sent by the JXTA Search hub on behalf of the searching client.

Unfortunately, the JXTA Search project appears to have halted development. There has been no code development since early 2002, and the code is no longer compatible with the current release version of JXTA. We were unable to run a prototype of the JXTA Search system successfully, and activity on the JXTA Search developers' mailing list suggests that others have had similar difficulties. We conclude that the project is not currently in an operational state, and there do not appear to be any current attempts to maintain the project.

4.4.4 JuxtaProse

JuxtaProse is a JXTA-based application whose goal is to provide "web / discussion content sharing the discussion application allows creating, sharing, viewing, and replying to HTML documents" [1]. JuxtaProse provides a user-friendly GUI which allows users to post messages or reply to messages, and post HTML documents to be shared with others. A search facility allows users to search for particular documents or messages. The JuxtaProse application builds on the JXTA Content Management System (CMS) described in Section 4.4.2.

JuxtaProse demonstrates the use of content advertisements through the CMS service. While its intent is to share HTML documents as well as other plain-text messages, it can easily be adapted to manage other resources such as files, in this case advertisements for components. Perhaps the most straightforward way to do so is to impose a structure to the text messages shared in a particular JuxtaProse community. Either plain text or

HTML documents describing shared resources can be created and shared, effectively serving as resource advertisements. JuxtaProse provides native support for classifying shared documents, including a "self-organizing classification system" under development, which could be utilized in organizing the file collection.

By developing a structured text representation of each component and then sharing those documents in JuxtaProse, a simple component sharing network was created (3). Components can be described using the previously defined metadata attributes, although JuxtaProse does not currently provide a very useful search facility (subject to the same limitations as the JXTA CMS service, i.e. it cannot search in the metadata or content of resources being shared). Components are classified by using the native JuxtaProse keyword classification system, and by "threading" the messages as in a discussion, with separate threads and sub-threads each containing categories of components. This creates a hierarchical and visually intuitive categorization of the components. Currently JuxtaProse does not hierarchically structure any shared HTML documents.

The JuxtaProse prototype imposes upon the user to correctly specify components in an arbitrarily chosen message format, and to not share incorrectly formatted documents or other text messages. An adaptor, message template, or standalone tool could be developed that would translate user-input component descriptions into the correct document format, to be shared in JuxtaProse. Alternately, JuxtaProse could itself be modified to share XML component advertisements directly rather than take the additional step of translating advertisements to/from HTML or text documents. In this manner, the approach would be very similar to that taken in U-P2P, where an XML schema was developed and used to describe component advertisements.

In addition to being a successful prototype bed for a component repository, JuxtaProse shows much promise in its demonstration of the abilities of a JXTA peer-to-peer network. The developers of JuxtaProse indicate that work is ongoing on other features such as adaptive searching, automatic classification, and content mirroring, all of which would benefit a file-sharing repository application.

Figure 3: The JuxtaProse application used to share components

5 Issues And Future Work

The development of prototype component repositories in U-P2P and JXTA illustrate some of the current limitations of their respective peer-to-peer frameworks. The U-P2P repository demonstrates the ease of which a component community can be defined and populated, using XML to create resource advertisements and using a browser-based interface to manage the repository. Its current underlying network structure is based on a central registration server and is thus subject to limitations on speed and scalability. Conversely, the JXTA core layer is flexible enough to support a wide variety of end-user applications and a variety of network and peer configurations. JXTA has a strong development community and has much support in the research community. However, the application-level projects are still in early development and some, like JXTA Search, are no longer actively maintained. While JuxtaProse comes close to delivering an applica-

Developing other services, such as that similar to CodeBroker [18] would also be important. Features such as integrated discussion forums, user-to-user messaging, and the ability to access network resources directly from within other applications, are all features which would increase the value of the network.

6 Conclusions

We have developed prototypes of a component repository based on a peer-to-peer network framework with the following characteristics:

- is standards-based (XML)
- features an easily-extensible ontology for component classification
- high specificity in search queries for efficient search (in U-P2P, JXTA support is forthcoming)
- distributed file storage and access across multiple peers

The use of a peer-to-peer network is designed to overcome many of the limitations of traditional client-server frameworks including limited repository sizes, the need to search and submit between repositories for maximum product exposure, inability to perform specific searches, etc. 6 summarizes the major advantages that peer-to-peer network file collections may have over standalone collections.

These results can be generalized to not only include software components but any type of resource (music, video, or even physical resources, personnel, etc.). A high-level classification using a set of metadata allows for more efficient resource management as well as increased accuracy in search results. Both U-P2P and provide the ability to define rich resource identification using XML (using either a custom XML schema or by a standard protocol, such as Query Routing Protocol).

It is clear from developing the prototypes in U-P2P and JXTA that each framework has its own set of strengths and weaknesses. U-P2P excels in creating a user-friendly interface which allows an end-user to quickly define metadata for any resource, and create a community for sharing resources, without the need for hard-coding a specialized application. U-P2P currently lacks a robust peer-to-peer implementation underneath the user interface, and is designed to be adapted to run on top of an existing peer sharing network. JXTA provides a solid core of low-level peer-to-peer services and protocols, allowing for robust network communication among peers. Its applications are numerous and varied, but no current JXTA application can be used for sharing of resources defined by user-specified metadata. The closest match, the JXTA Search service, is no longer under active development.

It is apparent that the two frameworks are complementary. U-P2P provides many services equivalent to JXTA Search. A version of U-P2P adapted to run on JXTA would provide an interface to allow end-users to define their own communities to share and find any type of resource on a robust and continually growing peer-to-peer network. The development of customizable content-sharing applications with a minimum of end-user effort is a feature not currently available in popular peer-to-peer frameworks. Development in this area would be tremendously beneficial to the community of peer-to-peer network users.

6.1 Acknowledgements and Author Biographies

Acknowledgements

This work is funded by the Carleton University Foundry Program.

About the Authors

Kevin Lam graduated in 2000 from Carleton's Computer Systems Engineering program (with high distinction), after which he gained two years of industry experience at Mitel working on real-time embedded software for telephony applications. He is currently working on a Masters Thesis in the area of applied data mining.

Dr. Babak Esfandiari is an assistant professor at the Department of Systems and Computer Engineering at Carleton University. He obtained his Ph.D. in Computer Science from

Standalone System Limitation	Improvement
Limited individual repository size	Individual repositories become peers in a much larger network. The entire collective repository is a superset of every collection, accessible from any single peer
No interoperability between repositories	Each individual repository conforms to the same XML standard for component advertisement, making it easy to move resources from one peer to another. Extensible, for future enhancement.
Duplication of resources across repositories	Made easier to handle since one search query returns all duplicate entries at once, which makes it easier to detect (and possibly filter).
No standard user interface	All repositories, regardless of individual characteristics, are accessible from any peer which may be running a client GUI
Varying or limited expressions of suitability	Ranking/rating system can be built into XML advertisement and supported by all peers; to some extent file duplication between client peers is a sign of popularity
Limited ability for users to interact with others, collaborate, share resources	Users become direct peers in the network. They can make use of instant-messaging features of P2P platforms, future work may allow tools to include embedded access to peer network
After finding candidate components, developers must examine them all for suitability; limited search options	Ontology-based, detailed and extensible metadata allows components to be specified and subsequently queried such as to return fewer, but more precise search results. The results obtained (if any) are more likely to match the developer's needs.

Table 6: Advantages of peer-to-peer networked component repositories over standalone ones

the University of Montpellier, France, in 1997. Prior to joining Carleton University in 1999, he was a software engineer in the Department of Strategic Technologies at Mitel Corporation.

References

[1] JuxtaProse Project Home Page. *http://juxtaprose.jxta.org/servlets/ProjectHome*.

[2] JXTA Content Management System FAQ. *http://cms.jxta.org/cms_faq.html*.

[3] JXTA Projects in Domain Website. *http://www.jxta.org/servlets/DomainProjects*.

[4] Project JXTA. *http://www.jxta.org/*.

[5] Universal Peer-to-Peer. *http://u-p2p.sourceforge.net/*.

[6] N. Arthorne A. Mukherjee, B. Esfandiari. U-p2p: A peer-to-peer system for description and discovery of resource-sharing communities. In *ICDCS 2002 IEEE Workshop Proceedings of Resource Sharing in Massively Distributed Systems (RESH 02)*, pages 701–706. IEEE Computer Society Publisher, 2002.

[7] Regina M. M. Braga, Marta Mattoso, and Cludia M. L. Werner. The use of mediation and ontology technologies for software component information retrieval. In *Proceedings of the 2001 symposium on Software reusability*, pages 19–28. ACM Press, 2001.

[8] Stuart M. Charters, Claire Knight, Nigel Thomas, and Malcolm Munro. Visualisation for informed decision making; from code to components. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 765–772. ACM Press, 2002.

[9] Wei-Tek Tsai Ci, J.X. Distributed component hub for reusable software components management. *Computer Software and Applications Conference*, 2000.

[10] S.Saroiu; P. K. Gummadi; S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proceedings of Multimedia Computing and Networking*, pages 1–15, 2002.

[11] Luqi Guo, J. A survey of software reuse repositories. Electronic Notes in Theoretical Computer Science, Volume 25, 2000.

[12] J.Cha; Y. Yang; M.Song; H.Kim. Design and implementation of component repository for supporting the component based development process. *Systems, Man, and Cybernetics, 2001 IEEE International Conference, vol. 2*, pages 736–739, 2001.

[13] E.J.; Sudha Ponnusamy P.; Wong E.B.; Mehandjiska D Meling, R.; Montgomery. Storing and retrieving software components: a component description manager. *Australian Software Engineering Conference*, pages 1–8, 2000.

[14] A. Mukherjee N. Arthorne, B. Esfandiari. U-P2P: A Peer-to-Peer Framework for Universal Resource Sharing and Discovery. *USENIX'03 (FREENIX Track) to appear*, 2003.

[15] O. Dikenelli R. Erdur. Data management issues in electronic commerce: A multiagent system infrastructure for software component market-place: an ontological perspective. *ACM SIGMOD Record*, pages 55–60, March 2002.

[16] Y. Maarek R.Helm. Integrating information retrieval and domain specific approaches for browsing and retrieval in object-oriented class libraries. *Conference proceedings on Object-oriented programming systems, languages, and applications*, pages 47–60, November 1991.

[17] Steve Waterhouse. JXTA Search: Distributed Search for Distributed Networks. *http://search.jxta.org/JXTAsearch.pdf*.

[18] G. Fischer Y. Ye. Technical papers: software presentation: Supporting reuse by delivering task-relevant and personalized information. *Proceedings of the 24th International Conference on Software Engineering*, pages 513–522, May 2002.

[19] L.; Prabhakaran B.; Bastani F.B.; Linn J. Yen, I.-L.; Khan. An on-line repository for embedded software. *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pages 1–5, November 2001.