UNIVERSITY OF CALGARY

Using a Peer-to-Peer Architecture to Support Distributed Software Development

by

Seth Bowen

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

DECEMBER, 2003

© Seth Bowen 2003

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled “Using a Peer-to-Peer Architecture to Support Distributed Software Development” submitted by Seth Bowen in partial fulfillment of the requirements for the degree of Master of Science.

_____

Supervisor, Dr. Frank Oliver Maurer, Department of Computer Science

_____

Dr. Günther Ruhe, Department of Computer Science

_____

Dr. Mike W. Chiasson, “Internal” External, Haskayne School of Business

_____

Date

ABSTRACT

A great deal of software is developed by teams, and these teams will often have members who work from different locations. Distributed software development tools facilitate the collaboration of these non co-located members and teams. These tools usually rely on a client-server architecture, which means that teams have to relinquish control of their intellectual property (IP) that is used for collaboration, because data is stored in a central location. The peer-to-peer (P2P) architecture allows a team to flexibly manage its IP when working temporarily with other teams in a virtual enterprise environment. This paper includes the motivation for this research, a discussion of the design issues, and a description of the tool that was implemented using Java and the P2P technology JXTA. A qualitative analysis was also done to assess whether the goals were reached, and to provide a comparison between MASE P2P and other distributed software development tools.

ACKNOWLEDGEMENTS

The author would like to thank Frank Maurer for providing invaluable feedback and suggestions during this research work, and for his comments during the editing of this paper.

DEDICATION

I dedicate this paper to my mom, dad, and two sisters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

API	Application Program Interface
ARPANET	Advanced Research Projects Agency Network
CVS	Concurrent Versions System
DBMS	Database Management System
DNS	Domain Name Service
DoD	Department of Defence
EAR	Enterprise Archive
EJB	Enterprise Java Bean
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Intellectual Property
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java Micro Edition
JVM	Java Virtual Machine
JNDI	Java Naming Directory Interface
JSP	Java Server Pages
JXTA	Juxtaposition
LAN	Local Area Network
MASE	MILOS Agile Software Engineering
MILOS	Minimally Invasive Long-Term Organizational Support
MSA	Module Specification Advertisement
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
PGA	Peer Group Advertisement
QRP	Query Routing Protocol
RIAA	Recording Industry Association of America
RMI	Remote Method Invocation
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UI	User Interface
URN	Uniform Resource Name
WWW	World Wide Web
XML	Extensible Markup Language
XP	Extreme Programming

1. INTRODUCTION

Software development is usually a team effort. The development of a project requires people with a wide variety of abilities, such as skills in management, design, and implementation. The people required for the completion of a project can not always be found who work in the same organization, or even location. A temporary alliance of people who generally work from different locations (i.e., are non co-colocated) is known as a virtual team [Tan2000]. Likewise, a virtual corporation or enterprise is when several teams from different organizations work together (usually temporarily) for some purpose [Kötting1999]. From a development perspective, a piece of software is typically decomposed so that the complexity of the system can be more easily managed. This modularization of software leads naturally to distinct components with well-defined functionality that can be implemented by separate teams. For example, one team could be in charge of the business logic and another in charge of the presentation layer. There are also the familiar divisions of labour between developers and marketers, or technical developers and writers. Some of the benefits of forming a virtual corporation for the development of a project are that it allows for the use of a wide range of specialized resources, and expenses (e.g., travel, leasing) can be reduced. In other words, it may be more cost-effective for an organization to outsource a component rather than developing it in-house.

In a virtual corporation, teams may wish to maintain some level of independence for several reasons: 1) they have a preference for how to do something (e.g., organize or manage their group), and 2) they want to protect their internal information. For example, teams working together on a software project may prefer to develop software within their group using a preferred process. There are many different approaches to developing software, from the more traditional (e.g., Waterfall Method [Royce1970]), to the more recent agile processes (e.g., Extreme Programming [Beck2000], Scrum [Schwaber2002]), but no process is ideal for every situation or organization. A company’s internal

information is extremely important in the information technology industry. As Kemp et al. observe, “Knowledge is to the Information Age as oil was to the Industrial Age. Today, the main asset of production is intellectual capital as opposed to the tangible assets that previously drove manufacturing-based markets.” [Kemp2001]. Nowadays, being able to protect ones intellectual property (IP) is more important than ever because of the ease in which people can gain access to and distribute information on the Internet. Companies certainly understand the importance of protecting IP, as demonstrated by the increasing interest in making software and operating systems more secure. IP can encompass many types of information, such as project information (e.g., workflow breakdown, the people working on a project), an algorithm used to find genes in a genetic code, or a method of cataloguing WWW (World Wide Web; hereafter referred to as Web) pages on the Internet. IP could even be considered in a broader sense to include third-party information that an organization depends upon and is expected to protect, such as people’s medical records. However, the wish for independence must be balanced with the ability to collaborate because communication is required in any team or corporation for the coordination and management of work and artefacts.

A project being developed by more than one person requires collaboration. Collaboration includes communication and the sharing of information, as well as cooperation. Project tracking is one form of collaboration that is beneficial for the success of project because it allows team members to improve the estimates of deliverables as the project progresses. Projects can be tracked using a workflow breakdown structure of the tasks required for the completion of the project. Another form of collaboration is if someone makes available a profile of his or her contact information and skills, which aids others in recruiting the appropriate people into virtual teams and corporations based on their abilities. Tools provide support in these and other types of collaboration in software development projects. As well, tools are useful in the automation of repetitive and time-consuming tasks. Virtual teams and corporations can also benefit from the use of collaborative tools because the amount of computer network connectivity around the

world provides an extremely large set of resources that can be utilized for the successful completion of a project.

Most virtual organizations (i.e., virtual teams and corporations) make use of tools that are based on the client-server architecture, where generally powerful computers called servers are responsible for providing services to less-powerful workstations called clients. The peer-to-peer (P2P) architecture is a more flexible architecture, where each computer is able to have equivalent responsibilities, and so they are referred to as peers. This means that a P2P network allows teams to share information without having to store their IP on a centralized server that is outside of their control. Thus, a P2P architecture allows teams working on distributed software development projects to balance the conflicting goals of maximizing control of their IP, and maximizing their level of collaboration.

The control of IP and collaboration are at different ends of a spectrum. How these two conflicting goals are best balanced is dependent upon each given situation, and so a flexible solution is required. The P2P architecture allows teams to have more autonomy and control of their information because it is possible to have a de-centralized organization of information in the network. This paper addresses the difficulty in coordination of a multi-team project when groups wish to retain control of their intellectual property. A P2P tool was developed that allows teams to choose a point between maximizing the control of their IP, and sharing certain IP (member profiles and workflow breakdown structures) with other teams.

The remainder of this paper is organized as follows. Following this introduction is an outline of the objectives for this research. Chapter 2 includes an analysis of the related work. The requirements for the MASE P2P tool are outlined in chapter 3, along with the motivation for why the tool is needed, and an example scenario of when the tool could be used in real-life. The design issues that were considered are addressed in chapter 4. Chapter 5 includes a discussion of the implementation details. Some lessons learned

during the development of the tool are discussed in chapter 6. In chapter 7, an example of the MASE P2P tool being used in a real scenario is described. A qualitative analysis of whether the goals of this paper and the requirements of the tool were achieved is found in chapter 8. The conclusions of this research, and possible future work, are found within chapters 9 and 10, respectively.

1.1 Objectives

The objectives of this research were fourfold:

1. *To develop a flexible solution that allows teams within virtual corporations to balance the control and sharing of their IP in distributed software development projects*. A flexible solution was needed so that a team could choose based on its preference how much IP it wishes to share. Current distributed software development tools do not support maximal control of IP. IP sharing is a form of collaboration that can include numerous activities, and so team-member profile visibility and project tracking were selected as representative activities.
2. *To examine the design issues for a tool, which is based on a P2P architecture*. An overview of the requirements for the tool precedes the discussion of the design issues. Since the tool is based on a P2P architecture, which is a distributed architecture, several of the design issues are applicable to any distributed application.
3. *To implement a tool as a proof-of-concept*. A working implementation demonstrates that it is possible to build and use a P2P tool that allows teams to have control of their IP and collaborate in a distributed software development project. The implementation is distributed under the MIT open source license [MIT2003], and so there is the opportunity for other researchers and developers to build upon the tool, which will hopefully foster new ideas in this research area. The implementation of the tool involved adding an extension to an existing client-server distributed software development tool called MASE.

4. *To do a qualitative analysis of the approach to assess its merit and address any issues*. The analysis includes an assessment of whether the MASE P2P tool improves upon existing tools that are available today.

2. RELATED WORK

This section provides a background and analysis of the relevant literature related to the work described in this paper. It begins with an outline of distributed software development and the tools used to support distributed software development. This is followed by an analysis of two different types of distributed computing, centralized and decentralized.

2.1 Distributing Software Development

The proliferation of computer networks around the world allows people to collaborate on software projects while working in different locations (i.e., being non co-located). These people may choose to form a virtual team, which is a temporary network or alliance of people. Similarly, a virtual cooperation or enterprise is a temporary network of independent organizations [Byrne1993]. Virtual teams and corporations are both affiliations that form in order to facilitate work among groups of people with a common objective.

Virtual teams working on open-source projects have produced software that is in common use today, such as the Apache Web Server [Apache2003], Linux operating system [Linux2003], and Perl programming language [Perl2003]. In fact, over 62% of Web servers were running Apache server software in January 2003 [Netcraft2003]. Anyone can download and freely use any of these applications according to the open-source license agreements. The people who work on the projects are often from several different countries, and so the work environment is generally highly distributed. There are usually few strict roles within the teams, and so developers have the freedom to do the type of work they enjoy most, such as design, implementation, testing, or maintenance. SourceForge.net is a website that manages a repository and configuration management services for over 70 thousand open-source software projects [SourceForge2003].

Projects that are developed in a globally distributed manner (i.e., global software development) are becoming increasingly popular [Carmel2001]. There are several reasons why distributed software development may be beneficial for certain projects ([Mockus2001a]):

1. limited on-site experience,
2. wanting to get closer to customers to provide location-specific expertise,
3. cheaper labour, and
4. the ability to do round-the-clock development.

In addition to the potential benefits of distributed software development, there are some difficulties that arise because of the potentially large distances separating the participants:

1. loss of information when not meeting face-to-face,
2. cultural differences,
3. time zone differences,
4. coordination and organizational problems,
5. security issues, and
6. technological differences.

The first three difficulties can hinder the communication in the group. Team members must be able to communicate effectively in any software project. Project managers have to be able to communicate with the team members in order keep up to date on the status of tasks and to help the developers deal with problems they may encounter. Also, there must be communication between teams when they are working on components that are dependent on each other. As well, teams can share their experiences and knowledge with other teams, which provides a benefit to everyone. Deficiencies within the communication of a group can cause delays and have a deleterious effect on the success of the project.

First, there is a loss of information when people communicate in a distributed environment because the people cannot speak face-to-face. This loss of information could have an adverse effect on the social environment. Some face-to-face communication is often necessary to allow team members to get to know one another and develop trust [Dutoit2001]. For example, there may be a lack of trust and willingness to communicate openly if there is a fear that sharing expertise may lead to being replaced [Mockus2001a]. Although video conferencing is a technology that provides both visual and auditory information, there is nonetheless a loss of data because of there being less resolution compared to face-face communication. In other words, all distributed forms of communication, including video conferencing, have a lower bandwidth than face-to-face communication. That being said, it is possible that the loss of information may not have a noticeable effect on the exchange between the parties, or on the success of the project. However, this question will not be addressed here because it is outside the scope of this paper.

Second, people who are geographically separated by great distances may be part of different cultures, which can also widen the barriers of communication. Understandably, language differences between cultures can cause a tremendous barrier to communication. The participants should be aware of the differences in cultures, otherwise there could be unnecessary misunderstandings [Damian2002].

Third, if people are working in different time zones, then it may be highly inconvenient for people to speak synchronously (e.g., using video conferencing, telephone, or instant messaging) if there is a significant time difference. An alternative is to rely more on asynchronous means of communication. For example, e-mail is a technology that is easy to use and has the added benefit of serving as a historical reference.

Fourth, the division of information may lead to organization and coordination problems. There could be unnecessary coordination overhead if the labour in a project is partitioned

poorly. For example, if there are preventable dependencies between work products from different sites. Or, two developers from different sites may not be able to work effectively because they were not able to meet and build trust and a working relationship in person. Organizational problems may also result when teams have to give up their independence in order to effectively collaborate with one another.

Fifth, it is more complicated to enforce security in a distributed rather than local setting. The only way someone can access information on a stand-alone machine that is not connected to the Internet is to physically be at the console and be able to log in. No additional security features beyond the basic password-protected user accounts within the operating system are required. However, if several computers are connected together in a network, then users have the opportunity to connect remotely, and automatically, which adds more complexity for enforcing security.

Sixth, if different network protocols or software technologies are used at the sites then incompatibilities will result in miscommunication. For communication to be effective, the rules of the communication protocol must be understood by both parties. The standardization of networking protocols (e.g., TCP/IP [Transmission Control Protocol/Internet Protocol]), and application protocols (e.g., HTTP [Hypertext Transfer Protocol], XML [Extensible Markup Language]), does help applications to communicate with one another even when they are being run on different platforms (i.e., operating systems).

Distributed software development is a viable option for many software projects, although the benefits need to be considered along with the drawbacks. Teams need to realize that measures must be put in place in order to deal with the adverse effects a distributed work environment can have on the communication (and possibly success) of a project.

2.2 Tool Support for Distributed Software Development

Tools are indispensable when doing distributed software development. One of the main goals of tools is for supporting communication, such as making communication easier to do, more efficient, and richer. There is an understandable overlap between some of the tools that are used for distributed software development and the tools that are used in everyday use by non-developers. For example, the telephone and e-mail are both common tools that software developers will certainly use when communicating with one another. As well, an increasingly popular tool is instant messaging (e.g., MSN Messenger), which allows people to communicate in real-time by typing, speaking, using a video-feed, or even using a shared whiteboard.

Electronic means of communication even provide some benefits over face-to-face communication. People using these tools can communicate asynchronously (e.g., e-mail, mailing lists), which means that regardless of people's schedules people can contact one another even if they are not working at the same time. Mailing lists are a form of asynchronous communication that is the predominant method of communication used in open-source projects [Crowston2002]. A mailing list is an archive of messages and replies that are posted regarding different topics. Although it is important that knowledge is stored in some form, it is of no use unless the information can be efficiently retrieved. When a mailing list is combined with search capabilities, it can be a powerful textual experience base. For example, Google (http://www.google.com), a popular search engine, allows searches on more than 800 million messages dating back to 1981 from the UseNet discussion groups [Google2003b]. Although mailing lists are only textual archives, they are easier to maintain compared to some other types of records (e.g., audio and video) because information can be automatically extracted relatively easily.

2.2.1 Process Support for Distributed Software Development

In addition to tools providing support for communication, support is also needed for the management of the tasks, work products, and the coordination within and between teams.

These types of support are especially important when projects are large and complex. A typical set-up for process support in a distributed project involves the storage of the project's meta-data and artefacts on a centralized machine so that the members of the team have access to the data. This situation results in independent teams in a virtual corporation having to relinquish control of the data that is used during the project development. This drawback with the current distributed software development process support tools is one of the main motivations for the research addressed in this paper, and it is addressed in more detail in section 3.1.

Distributed collaborative development tools may include support for process modelling, enactment, and collaboration among developers. Most work is goal and process oriented, which means that team members have specific tasks they do, and they follow certain processes to accomplish these tasks [Kötting1999]. A process model is derived by distilling the processes for a project into their essential elements. The task breakdown (or workflow) can even be used as a model in future projects. Process enactment involves providing support for the real-time development of the tasks within a project. This may involve providing information about the status of the project, or aiding in the coordination of work among the team members; for example, providing a mechanism to build (i.e., compile) source code from several locations. Microsoft Project is an example of a tool that supports project planning and tracking for distributed teams [Project2003]. Support for collaboration may include aid for both synchronous and asynchronous communication. Asynchronous communication is an important form of communication when people are working remotely because of time zone differences.

Configuration management is an integral part of software development that deals with the management of artefacts in a project. Version management is one aspect of configuration management that allows developers to permanently save versions of artefacts. A user then has the option of going back and restoring an earlier version of an artefact, which may be necessary if changes need to be undone. For example, if some program

functionality is broken when new source code is integrated into a project, then the clean source code can be retrieved from a previous version. Configuration management also includes the synchronization of several instances of an artefact, for which there are two approaches: file locking and merging. Locking means that only one user can edit an instance, or portion thereof depending on the level of granularity, at one time. Merging usually has to be done manually at the time of synchronization, but it has the advantage that work can be done concurrently on several instances of an artefact. That being said, the merge process may be minimal if different sections of the artefact were affected. Merging is certainly less restrictive than locking, but it requires diligence when two instances of an artefact are integrated so that the merging is done correctly. CVS (Concurrent Versions System) is an example of a version management tool that uses merging and allows for remote access to file repositories [CVS2003].

Tools that provide process support often contain security features. When teams are distributed, there are some files that are accessible by everyone, and other pieces of information that may be restricted to a few individuals. For example, some technical white papers may need to be accessible by everyone, while task assignments for a developer may only need to be accessible by a manager and the developer who is assigned to the tasks.

There is an active research community that is involved in discovering better ways to support these areas of distributed software development through the use of tools. CHIME is a tool that aids in the understanding of tasks and work products in a project by using a 3D virtual environment [Dossick1999]. TUKAN is a collaborative tool that includes the following support for Extreme Programming (XP) projects: version management, a coordination system, and communication support [Schümmer2000]. MASE is another tool that facilitates agile development in distributed projects; it is described in detail in the next section.

2.2.1.1 MASE

MASE[1] is a Web-based tool that provides process support and knowledge management for virtual teams doing agile software development [Bowen2002a]. It includes features for doing project planning, execution, and monitoring. The tasks of a project are represented as a workflow, which is a tree-like structure of the task breakdown. An example of a task may be to write source code or do documentation. The workflow structure is dynamic because it can be altered at any time if changes are needed; for example, if tasks need to be added or information about tasks updated. This is an essential feature because in most software projects the requirements will change throughout the development of a product [Harker1992]. It is also important for managers and developers to know the status of the development so that tasks can be coordinated among the developers. The knowledge management portion includes developer profiles, which includes the developer's contact information and skills. A project manager can survey developer profiles to determine whether there is anyone with a specific skill available to do a task. For example, if a system written in Java was going to make use of a Web-based user interface (UI) then a developer with knowledge of Java and HTML (Hypertext Markup Language) would be sought. MASE also includes an experience base, which is a knowledge store for generic process models that can be used as templates in new projects.

Agile practices aim to control software development without overly defining the processes [Cockburn2002]. An HTML interface allows developers to plan their

---

[1] MASE was first known as MILOS, which is an acronym for Minimally Invasive Long-Term Organization Support for Software Development [Maurer2000]. This system was first implemented at the University of Kaiserslautern, Germany with a Java Swing GUI user interface, and made use of an object-oriented database. Development of a branch continued at the University of Calgary under the direction of Frank Maurer and his group [EBE2003]. The branch made use of an HTML user interface and a relational database. The agile support was then integrated into MILOS, and so its name was changed to MASE, which means MILOS for Agile Software Engineering.

development using a standard Web browser (e.g., Internet Explorer). In MASE, project development is planned, monitored, and managed using releases, iterations, user story cards, and tasks as explained in the XP process ([Beck2000], [XP2003]). Furthermore, pair programming can be initiated and conducted using Microsoft NetMeeting [NetMeeting2003]. In addition to MASE, there are several other open-source tools being developed at SourceForge.net that support agile software development, but most of them are only in the early stages of development (e.g., Agile Manager [AgileManager2003]). There are also commercial products available (e.g., VersionOne [VersionOne2003]). Agile practices provide extensive benefits to software development: the customer quickly gets business value for its investment, the requirements that inevitably change are embraced, and the development is tightly controlled without being overly defined.

2.3 Distributed Computing

Distributed computing involves connecting several machines together for a common purpose. Computers that interact with one another can be part of a configuration that is more centralized or decentralized. A centralized architecture tends to have information stored on a single machine, or several machines tightly connected in one location. The client-server architecture is an example of a centralized configuration. A server is usually a powerful computer that provides resources to clients, which tend to be less-powerful workstations. A decentralized architecture may consist of information that is spread out over multiple locations. An example of a decentralized layout is the DNS (Domain Name Service), where a hierarchy of machines is used to resolve the mapping of Web server domain names to Internet Protocol addresses. A decentralized set-up can exist without the support of a single company or person (i.e., for hardware or infrastructure). There are several important characteristics of these two architectures that can be grouped under benefits and drawbacks; however, it is important to remember that in a given situation some of these characteristics could end up being either advantageous or disadvantageous.

2.3.1 Benefits and Drawbacks of Centralization

There are beneficial characteristics of the centralized architecture. Many of these benefits translate to simplicity in the implementation and maintenance of systems using this architecture.

- *Searching on a single repository*. Centralization generally allows for complete and relatively easy searches because no coordination or communication is required between computers.
- *Data updates on a single repository*. Data updates are often simple because there is often just one repository. No synchronization is required because data is not duplicated.
- *Administration of a single repository*. Administrative duties may be straightforward because the information is only on one machine. Thus, security mechanisms, such as authentication and authorization privileges, do not have to be enforced across multiple machines. As well, if any code updates are needed then they only have to be done to one machine. The simplified security mechanisms implemented by the administrator translates to the users being able to more easily access information because they only need to adhere to one set of settings (e.g., firewall settings).

Centralization also has its drawbacks.

- *Single point of failure*. Information may be lost or become unavailable if it is stored on one machine and the machine crashes.
- *May not scale well*. If the number of users requesting resources from a server becomes too large then there may be lengthy delays due to congestion. Alleviating the congestion may require increasing the processing power of the machine or increasing the bandwidth of the connection to the machine.

2.3.2 Benefits and Drawbacks of Decentralization

A decentralized architecture has several characteristics that may be beneficial. These are listed and explained below:

- *Amount of information.* One of the major benefits of connecting several machines together is that there is opportunity to greatly increase the size of the repository of information. Much like in research, if a researcher only works by him/herself then there is little opportunity to make use of other people as resources, which could lead to fostering new ideas. The Internet can be considered the world's largest database because of all the Web pages that are available. The Google search engine has references to over 3.3 billion Web pages [Google2003a], and an estimate for September 2002 is that there are over 600 million users who use the Internet [Nua2003]. There is clearly a great deal of information available to users who have a connection to the Internet.
- *No single point of failure.* There is the potential to reduce the chance of a single point of failure. In fact, the Department of Defence (DoD) initially intended to use the Internet as a network of services for the military that would not suffer from a single point of failure in case there was an attack [Leiner2000]. If there is a failure at the one location where the information is stored then that information may become unavailable or it could be lost.
- *Load balancing.* There is the opportunity to distribute the network load to multiple machines in order to reduce the chance of traffic hot spots developing, which can lead to long delays due to congestion. For example, one server will not be bogged down with requests from clients for a popular file. Additionally, the network lines going to the server will not become congested.
- *Scalability.* Because of the ability for the load to be distributed in a decentralized network as more computers are added to the network, decentralized services tend to scale better than centralized services. Computers can usually be added to a network without the network buckling under the strain. That being said, the protocols being used in the network may require modifications in order for the

network to scale, as was experienced with the Gnutella network (see section 2.4.1 Applications).

There are the following drawbacks to decentralization.

- *Searching across several repositories.* Since information is not stored on one machine, a search may have to be done on several machines and then joined into one result. The complexity of the search will have an effect on the time to complete the search because of the overhead involved.
- *Data synchronization.* There is the possibility of having to do data synchronization over the network if there are duplicate copies of data in a decentralized architecture. Data synchronization is required if copies are to be kept up to date.
- *Complex administration.* Administrative duties become more complex as more machines need to be administered. For example, if security privileges encompass several machines.

2.4 Peer-to-Peer

Peer-to-peer (P2P) is both a social and technical idea [Lethin2003]. There is no agreement on a precise definition of P2P. As Barkai [2001] points out, however, it is sometimes desirable to not have a rigid definition when the interest in a research area is new, because new technologies often introduce surprises that may need to be accommodated. For this paper, P2P will be regarded as both a computational model, and a set of technical protocols.

The client-server model is the predominant model used in the Internet. This model consists of machines that spend most of their time serving information to other machines called clients, which in turn spend most of their time requesting information from servers. An example of the client-server model is a mail server that stores e-mail messages for

clients that login and retrieve their messages. Any messages that a client sends to another client must go through the mail server.

P2P was first defined to say what it is not--P2P is not about centralization. A P2P network can be completely decentralized. For example, the Gnutella network is a decentralized environment where users can share files directly with one another. P2P is a network-based computing style that does not depend on centralized servers and at the same time does not exclude them [Gong2001]. P2P is about dynamically discovering other peers and connecting regardless of whether a prior relationship exists. It allows people to interact over a network dynamically and directly. The roles of computers are blurred in P2P. The nodes do not have permanent roles (e.g., a server, client, or router), but rather, they can exhibit any number one of these behaviours as the need arises. In other words, the nodes (or peers) have equivalent capabilities, and can exchange information directly with one another. This differs from the client-server architecture, in which some computers are dedicated to serving others. In essence, the P2P model can be thought of as a more flexible model than the client-server model because the roles are less rigid.

An important point to remember is that although all peers are created equal in a P2P environment, the differences of the peers should be exploited for the benefit of the environment. For example, if a peer has relatively little processing power and/or a slow connection to the network then it should not be in charge of storing information about other peers (e.g., for searching), because it will hinder the performance in the network.

The P2P model is not new. The ARPANET (Advanced Research Projects Agency Network), which was the precursor to the Internet, came about at the end of 1969 and it can be considered a P2P network because it had four computers with similar roles connected together [Leiner2000]. The telephone system also works like a P2P system because users connect directly with one another.

It is difficult to clearly say whether one application or network is P2P, while another is not [Minar2001]. For example, Napster was a highly popular application that is regarded as being P2P; however, it is more of a brokered P2P service because there was a central server that handled search requests. A user could search for and identify other users with a desired file and then connect directly to download the file. Here is a question, if two servers send information to each other, is this not an example of P2P? It is in the strictest sense. However, P2P is more about using the edges of a network, which means using the computers that at one time were primarily client machines. A P2P network is one where every machine has the ability to adopt multiple roles. Although there is usually direct communication between peers, this is not always the case. For example, the SETI@home project is considered a P2P network. Resources from people's home computers are used to contribute to a common goal, although the peers do not communicate with one another directly. The debate on what constitutes a P2P application is an interesting one, but for the purposes of this paper it does not require further discussion.

In the past few years, there has been increasing interest in researching P2P networks. This research deals mostly with the technical aspects, such as evaluating the cost of doing searching in a P2P network [Portmann2001], or adapting the network to make the best use of peers with different abilities [Renesse2002]. A great deal if this research is related to the performance of P2P networking, for example, improving the efficiency of discovery [Ripeanu2001] and searching [Aberer2002]. There are currently several protocols for P2P networks (e.g., Gnutella [Gnutella2003], Jabber [Jabber2003]) that could be compared using several factors, such as reliability, performance, or services offered. Some researchers are even investigating how P2P networks affect social issues, such as the trust of digital documents [Mont2001].

2.4.1 Applications

The success of the P2P architecture will likely be dependent upon the applications that arise from the technology. Building P2P networks and applications requires addressing some of the same technical issues as in any distributed computing applications, such as naming, discover, security, etc. There are certain types of applications that lend themselves well to the P2P idea [Gong2002]:

- distributed directory systems,
- network file systems that support disconnected computing,
- parallel systems to make use of CPU cycles,
- distributed messaging and e-mail systems,
- fault-tolerant systems,
- collaboration,
- distributed computing, and
- content sharing.

Kazaa is a popular P2P file-sharing application. At any one time, you may find over 4 million users logged on, and there could be 6 petabytes (i.e., 10^{15}) of data that is freely available. Several applications are geared towards collaboration, such as Groove [Groove2003], and Jabber [Jabber2003], which is an open protocol for the real-time exchange of messages. Some are geared towards the long-term storage of information (e.g., Free Haven [FreeHaven2003], and preventing censorship of information (e.g., Freenet [Freenet2003], Publius [Publius2003]). Now, some types of server-mediated P2P services are increasingly popular, such as instant messaging programs (e.g., MSN Messenger [Messenger2003]), where users sign in to a server, but then communicate and share files directly.

The popularity of P2P was first attributed to Napster, a music-sharing service. Napster was estimated to have close to 60 million users of their service [Costello2003]. Understandably, the big record labels did not like the idea that people could download

CD-quality music for free. The Napster service was dependent upon a centralized store for searching, which happened to be managed by the company, and so the courts could directly assign blame to the organization. Once Napster was shutdown by the courts, other P2P networks and applications that did not require centralized components became prominent, such as Gnutella [Kan2001] and Kazaa [Kazaa2003]. In addition, the Gnutella and Kazaa networks allow any types of files to be shared (e.g., music, video, or software applications). The Gnutella network is not owned or controlled by any kind of company, and so music companies and the RIAA (Recording Industry Association of America) cannot target an organization, but would have to go after individual users. It is important to keep in mind that P2P systems are not used solely for illegal music swapping.

The Gnutella network is a completely decentralized file-sharing network that was created by Justin Frankel and Tom Pepper working for NullSoft, which was owned by American Online (AOL) in March 2000 [Kan2001]. A client program was released, but then pulled hours later due to a reasonable fear of legal action for the creation of a network where users could potentially share copyrighted material. People had already downloaded the program and so developers eventually reverse engineered the protocol and created their own open-source protocol. There were scalability problems with an early version of the protocol because all computers in the network were considered equal. This was a problem because each search request was broadcasted at each machine, which meant that the network was inundated with traffic whenever a search was performed. Additionally, because there was no hierarchy in the network (i.e., it was flat), the less powerful machines were bogged down with requests, and this ended up slowing down the entire network. A more mature release of the Gnutella protocol makes use of a hierarchy of roles where ultra peers that have high bandwidth connections shield leaf peers from giving responses. This approach is also used in other P2P protocols, such as the FastTrack network [Kazaa2003]. A decentralized network is in theory very scalable if peers are willing to share the load.

2.4.2 Benefits

A P2P environment has several characteristics that may be beneficial depending on the requirements of a system. Albeit, while these features are characteristics of the P2P environment, many of these features can also be introduced into a typical client-server setup if the effort is put into implementing these features. The benefits have been divided into social and technical benefits.

2.4.2.1 Social Benefits

1. *Huge knowledge store*. Users can benefit from the contributions of a large number of peers. A P2P network is generally very scalable and so can easily support a very large amount of information.
2. *Real-time information*. A great deal of Web content is invisible to the current search engines. This is in part due to the fact that information on the Web is constantly changing, and also because a great deal of content on the Internet is not found in static Web pages but is generated dynamically. Dynamic content is becoming increasingly popular because Web sites often want to customize a site to individual users, or display session information when a user visits several pages in a site. The dynamic content on the Internet means that search engines will reference obsolete links, or cache outdated information. Search engines cannot keep up with the dynamic content that arises on the Internet, and the proportion of documents on the Internet that are accessible via indexing search engines is shrinking ([Lawrence1998] and [Lawrence1999]). A P2P environment does not normally rely on search engines for cataloguing static pages, but rather will use indexes that are maintained and published dynamically.
3. *Anonymity*. Users can remain anonymous. It is difficult to determine who produced a file and who downloaded it in a highly distributed network.
4. *Deniability*. Some P2P services allow users to use their machine simply as part of a distributed store, but they are not aware of the information being stored and so the users can deny knowing what information is on his or her computer.

5. *Resistance to censorship*. The ad-hoc and dynamic nature of the network means that it can be impossible to censor information.
6. *Startup ease*. There is no need for special administrative or financial arrangements to starting a P2P network. On the other hand, a centralized service is more likely to mimic a centralized organization, which would require such arrangements [Balakrishnan2003].
7. *Availability*. A P2P system allows for incredible redundancy, which supports the long-term storage and access to information.
8. *Flexibility*. P2P allows for flexibility within the network. For example, new nodes (and data) can be introduced quickly and easily, which means the layout of the network may change spontaneously. The administrative overhead can be administered at the most appropriate hierarchical level.

2.4.2.2 Technical Benefits

1. *Massive scalability*. There is the opportunity for a malleable network structure that can support large numbers of new peers without the problem of one or more machines or the network being overloaded with the increase in requests and traffic.
2. *Redundancy*. The network will become more robust when nodes participate and store replicates of data on their own machines [Macedonia2000].
3. *Reliability*. There is the opportunity for increased reliability and fault tolerance in a P2P system. For example, an e-mail message could be sent directly to a client instead of going through a mail server, which may be overloaded at certain times.
4. *Short delays*. There is the potential for shorter delays when data is accessed over a P2P network because peers can retrieve data from a site that provides the shortest response time. The delay in accessing information in a network can be dependent upon the speed of the computers involved in the transaction, the congestion in the network, and the bandwidth of the line. A user wishing to retrieve data from a centralized site must have an adequately fast connection to access the information because there is only one source for the information. The connection speed is

related to both the amount of bandwidth on the line, and the distance the user is from the source. High-speed Internet connections are quite common in North America; in particular, as of January 2003, 53.6% of the Canadian online population were broadband users [Reid2003]. However, this number is likely lower in other parts of the world because of the infrastructure not being in place, or the service may be too expensive. Regardless, even if you have a high-speed connection to the Internet, bottlenecks can develop in the network that can cause uncomfortably long delays.

5. *Load balancing*. Load balancing can reduce traffic in the network. The effective bandwidth of a network can be increased when traffic is load-balanced, and so the peak load on the network is reduced [Lienhart2002].
6. *Distributed operations*. Operations that require complex computation can be distributed over several machines, which means several processors can share in the workload.
7. *Good use of bandwidth*. Bandwidth is currently over-utilized in some areas in the Internet because hot spots develop when sites become popular (e.g., Hotmail, Yahoo), while cools spots remain cool. The P2P architecture allows for bandwidth usage to be spread out instead of it being concentrated, because the traffic can be distributed across several machines.
8. *Strength of the network*. As the network grows, resources (e.g., storage space, cycles) are added and so the network becomes stronger.

2.4.3 Drawbacks

The P2P architecture has fewer drawbacks than advantages, but this is not to say that the P2P architecture should always be used because in a given situation the cost of the drawbacks may outweigh the benefits.

- *Management complexity*. Enforcing security policies or backup policies in a dynamic environment where peers (i.e., computers) can become online or offline at any time adds complexity to the system.

- *Synchronization*. Files that are replicated in the network may need to be synchronized when a file is changed. This requires propagating changes to peers. In other words, based on the requirements of an application, a mechanism might have to be put in place to ensure that a peer has the most recent version of a file.
- *Service availability*. Because of the dynamic nature of a P2P environment, services may become unavailable unexpectedly unless some provisions are made to ensure that services are available 24 hours a day and 365 days a year.
- *Difficulty to authenticate*. Authentication is often important in applications where certain privileges are needed in order to access information. The dynamic nature of P2P environments does not lend itself to easily tracing peers, such as through static IP addresses or registered domain names.
- *Search complexity*. Implementing a search across several machines is more difficult than on one machine.
- *Maintaining performance*. Although every peer has the opportunity to contribute resources to the network, peers will differ in how much processing power they have and the speed of their connections to the network. The network has to be able to adapt to make use of the best resources and not get bogged down with slow peers contributing too much.
- *Free-loaders*. The intent of a P2P environment is that users contribute to the network. However, unless peers are willing to cooperate in contributing information, the network will revert to a few contributors with many users who only download data. That being said, software development is different than file-sharing because the users are part of a team with a vested interest and so it could be argued that in this domain most users would be interested in contributing.
- *Trust*. If data is obtained from a peer that is not a primary source, then the issue of trust arises. Can a third party be trusted to provide the unadulterated copy of the original? The authenticity of the file can be verified with the use of cryptography. There is also the issue of trust with respect to the identity of the peer. Certificates

can be used to verify the identity of a peer; however this just means that the trust is transferred to the certificate authority.

2.4.4 Distributed Computing versus P2P

There is a slight difference between distributed computing and P2P; however, the difference does not have much of an impact on the work of this paper, and so it is not discussed in great detail. While distributed computing usually assumes platforms have equivalent computing components, P2P is more geared towards heterogeneity for things like uptime, computing capabilities, and bandwidth connectivity.

2.5 Security

Security is of tremendous importance when teams within a virtual enterprise (i.e., corporation) want to control their IP, and still collaborate. A virtual enterprise consists of teams that are working in different locations (i.e., a distributed environment), and so the following security issues arise: 1) data can be intercepted while it is being transferred, and 2) anyone has the opportunity to access resources because they are remotely available. Communication in a distributed environment means there are several more opportunities for people to eavesdrop and gain access to stored information than if access is limited to within a secure Local Area Network (LAN).

Security measures in distributed software development encompass several facets. Authentication is the verification that a user is in fact who he or she claims to be. Password verification or a method of certification can be used to help prevent impersonation. Authorization is used for restricting access to information to certain users. The security management for authorization can be centralized or decentralized. An example of a centralized layout is one where all of the authorization requests go through a central authority, such as when you log in using a Microsoft Passport. A decentralized layout could make use of a hierarchy of individual authorities. Data encryption can help prevent unauthorized people from viewing information. Information can be selectively

filtered so that only some users have access to information. All of these issues must be considered in order to have a secure system.

Security can be enforced at different layers in an application. A login mechanism is often used at the application layer, where a user name and password are required for authorization. Security mechanisms that are invisible to the user can be used at lower layers, such as secure sockets at the TPC/IP layer. Implementing security measures at different layers means that the functionality can be abstracted from implementation at higher-levels.

Establishing a group is a convenient method of enforcing security over several users or computers. For example, a LAN will sometimes hide the IP addresses of the machine inside the network from the outside so that internal machines cannot be targeted by hackers, and the behaviour of the machines cannot be tracked. When security restrictions are applied to a group, then a user will have the same security restrictions if he or she is a member of the group. Some form of authentication (e.g., logging in) is required in order to determine whether a user is a member of a group. Applying security privileges at the group level means that any number of users can be supported, and so it scales well. As well, if sub-groups are allowed then privileges can be easily inherited because of the hierarchical structure.

3. MASE P2P REQUIREMENTS

The purpose of this section is to address the requirements for the tool. The requirements came about from an analysis of the related work and the limitations of the existing distributed software development tools regarding data control.

MASE P2P was developed in order to show that virtual teams can make use of a tool based on a P2P architecture that provides flexible management of IP, where the IP is member profiles and workflow breakdown structures. Flexible control is needed so that the conflicting goals of IP control and collaboration can be decided on a per team basis based on each team's situation. The sharing of team member profiles allows teams to find people with the appropriate skills to work on a project. A workflow breakdown structure is used for progress tracking, which is useful when a virtual enterprise has several teams working on different components in one project.

MASE P2P can be thought of as a wrapper to the MASE system because the implementation is dependent upon MASE, but MASE is not dependent upon MASE P2P. The author felt this was an important decision because it means that the core MASE tool, which contains the vital functionality that teams require for distributed software development support, can be installed and run independently from MASE P2P.

This section begins with the motivation for why such a tool would be beneficial to distributed software development projects. The motivation serves as a foundation for why the requirements are deemed to be necessary. This is followed by a specific real-world example of a situation in which a project is being developed by two different organizations. Finally, the functional and non-functional requirements for supporting the control of IP and aiding collaboration are outlined for this tool, which is built on a P2P architecture.

3.1 Motivation for Using a P2P Architecture to Support Distributed Software Development

My motivation for using a P2P architecture to support the control of IP in a virtual enterprise is because the consequence of using a client-server architecture is that data is stored at a single location, which means that individual teams do not locally store their own data used in development, and thus they relinquish control of their IP. A virtual enterprise can be composed of several teams that include a mix of people with different backgrounds and interests who may want to maintain control of their IP because of the temporary nature of the affiliation.

The client-server architecture is dependent on a central authority, or organization, being responsible for the information that is stored on the server. There is only one store of data and so an organized group will need to assign an administrator, or administrators, to make decisions regarding the stored information. Some of the decisions may be related to deciding what information should be stored, where it should be stored, how it should be stored, and who should have access to it. It is common for different people to have different access rights for information that resides on one machine (i.e., a centralized store); however, it is uncommon for an administrator to not have access to all of the information because then it is difficult to maintain the data[2].

Most systems are not completely secure when it comes to preventing access to files, which may be problematic if there is sensitive data at stake, such as medical information. These deficiencies could be alleviated by clustering files on different machines and then restricting remote access. This leads to the P2P model of independent machines with

[2] For example, in the Software Engineering group at the University of Calgary, when moving data to a different server, it took almost twice as long, which ended up being an extra few hours, to transfer the data because some users had removed administrator access to some files [Stadel2003]. Nonetheless, the move was still possible because the administrator was still able to alter the users' file permissions in order to gain access to and copy the information.

similar responsibilities being able to connect with one another. The P2P model provides benefits because it is more suited for situations where people may have exclusive access to information, because each group can have complete autonomy.

If data is stored at one location then there is a dependency on a single machine. There are few things more frustrating than not being able to work on a file before a deadline because the network server is unavailable. Or, you may be waiting for an important e-mail and the mail server goes down and so you cannot access your e-mail. If data redundancy is introduced into a system then there is the possibility to have alternate machines providing access to information. This is similar to the idea of mirroring download sites around the world for software, which means that users will have the option of downloading information from the site that provides the shortest download time.

3.2 Example Scenario

An example is described in order to more concretely illustrate the value of using a P2P tool to support the management of IP in a virtual enterprise (see Figure 3-1).

Company A is developing a Web service that computes several different mathematical functions. Both the back-end functionality (i.e., business logic) needs to be implemented, as well as the user interface, or presentation layer, for the Web service. Company A decides to contract out the implementation of the math functions to company B, which is experienced in implementing efficient algorithms for mathematical functions. Company A will implement the Web service code, but it will only implement the service code for those math functions that have been implemented to date. The math functions are prioritized in terms of importance and will be implemented one by one. Company A wants to be the first to offer this service on the Internet and so it wants to make available the services as soon as possible. Thus, company A must have good knowledge of the

progress being made by company B and the interfaces that will allow their Web service code to connect to the code for the mathematical functions.

Company B has a reputation for designing and writing quality code. It has many customers and so can insist that it will release only the compiled binary code and not the source code unless an exorbitantly high price was paid by company A. This is one way that company B is able to protect its IP, because their customer cannot see how the functionality was implemented. In addition, company B may be able to get more work in the future because the customer cannot alter or enhance the existing functionality themselves. If company A wanted to make a change to the existing code, it would likely ask company B to make the change.

In terms of the actual development process, company B insists that it must have complete control of the development process because it uses different processes than company A. Company B feels that the process it is accustomed to using works well for their organization, and is most appropriate for the project. The company also wants to re-use some code and documents from past projects in the development of this project, but they do not want to release any of this information outside of the development team.

In this example, there is a division of the work in a project between two teams in different organizations. The work in one team is dependent upon the work being done by the other team, and so there must be some communication and collaboration between the two teams. For example, the team implementing the mathematical functions will need to provide progress reports for high-level tasks, which are needed for coordination between the teams. At the same time, the actual development process in the two teams will be different, and at least one of the teams wants to be able to protect the information it uses during the development of its part.

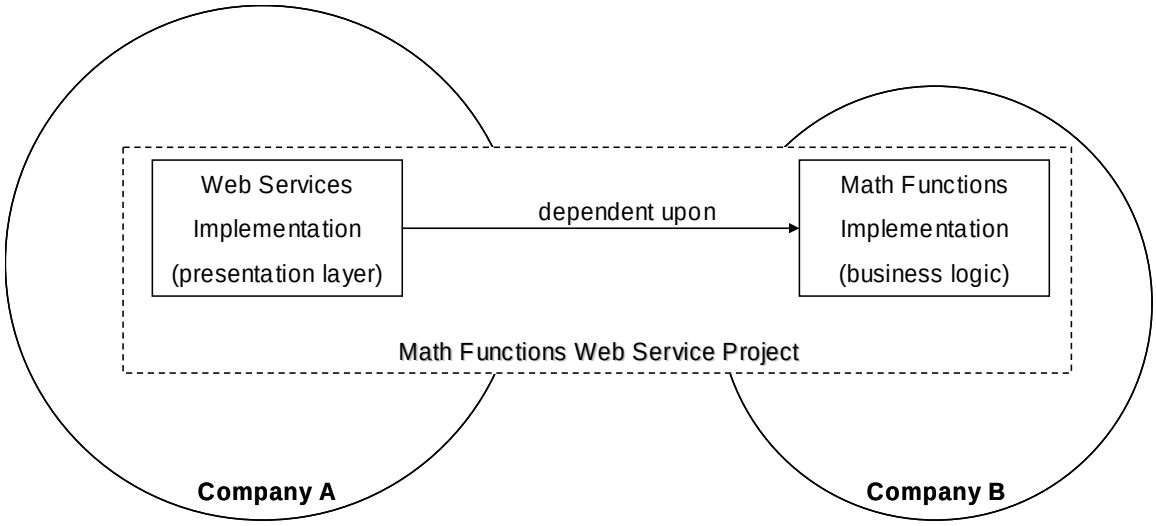

Figure 3-1. An example of a project being developed by two companies. A team from company A is implementing the Web services (i.e., presentation layer), and a team from company B is implementing the mathematical functions (i.e., business logic). The implementation of the Web services is dependent upon the implementation of the mathematical functions, and so project tracking is required in order to support a timely release of the system.

3.3 Distributed Tool Requirements

Since the tool will be used in a distributed environment, there are some requirements that are related to it being a distributed software development tool (inspired by [Lethin2003]). Since a P2P environment is a specific type of distributed environment, these requirements are equally relevant for a tool built on top of a P2P architecture.

- *Availability*. Users should be able to access the network at any time.
- *Authenticity*. Users will need to authenticate themselves by using a user name and password.
- *Access control*. Certain information will only be accessible by authorized users in order for groups to maintain control over the privacy of their data and IP.
- *Location independence*. A user should be able to retrieve information about how to contact groups and retrieve data regardless of the user's location.
- *Platform independence*. A user should be able to access information regardless of the platform (i.e., operating system) he or she is using. MASE can be used by

anyone who has a standard Web browser, which are available for most popular platforms (e.g., Windows, Mac OS, Linux), and an Internet connection. MASE P2P should also be useable by anyone who is running a popular platform.

3.4 Collaboration Requirements

The MASE P2P tool will support collaboration between teams, and so two types of collaboration are included: the sharing of team member profiles, and project tracking using a task workflow breakdown. The collaboration requires that appropriate information can be viewed, and also copied to a local machine within the team.

Knowledge sharing allows groups to become more efficient because they can obtain information that can be used to help solve problems. The knowledge must be both available and easily accessible for it to be worthwhile. In software projects, developers are a source of tacit knowledge. This tacit knowledge is especially important when agile development processes are used (e.g., XP), because there is less emphasis on documentation than the traditional methods of software development. For this reason, developers will often know a great deal of information about a project. For example, being able to search for developers by skill in other groups can help teams identify contacts that may be useful in helping to solve problems. Or, a team may need to find a developer with a specific skill that is not available in their team. A search query should be possible based on several criteria, such as name, skill, and group.

Project management includes several activities, including the planning and execution of development, and the management of work products and resources. The tool will specifically address the coordination of tasks through the use of project tracking. MASE represents the tasks in a project with a hierarchical workflow breakdown structure. This hierarchical structure is a convenient method of representing tasks because a task can be divided into sub-tasks to ease the difficulty in dealing its complexity. Furthermore, there is a clear and explicit association between tasks and so a task can easily be moved with

its associated child tasks, and there is the possibility for child tasks to inherit information from parent tasks.

3.5 IP Control Requirements

Teams should be able to control their data, which includes their IP. They should be able to store their data locally so that they do not have to relinquish control of this data to another team. Access privileges can be assigned to data in order to restrict access to certain teams.

A team is a mechanism of providing scope for people with similar interests or goals. The tool should support identifying different teams, and whether a user is a member of a team (i.e., authentication).

4. MASE P2P DESIGN

The purpose of this section is to describe the design decisions that were considered for MASE P2P. The author's personal experience using MASE, and an analysis of the related work (see chapter 2 Related Work) led to the issues discussed in this section. The majority of these issues are relevant for any distributed system, with the exception of the last. Most of these questions were taken from [Bowen2002b].

- Is there a solution that allows for the flexible management of IP, where knowledge is both protected and shared?
- What will be the roles in the network?
- What information is being considered?
- How is the information organized?
- How will the information be distributed?
- What type of data synchronization will be supported?
- What will be the search capabilities?

4.1 Protecting Intellectual Property

A group can only have complete control of its IP if it can store the information within the group. The P2P architecture allows teams to store their data locally, while at the same time making parts of it available to other teams. Teams have access to information based on the whether they have the appropriate authorization privileges. Users must be able to be authenticated in order that their identity and membership within a team can be confirmed.

It is important that a user's identity can be verified (i.e., authenticated) before the user is able to access information that may be private or confidential. In a distributed environment, users can be anonymous because they can join the network from an arbitrary machine. Although at some level an Internet Protocol address and port are

known because packets need to be sent to a specific machine, this information is often hidden from the user at the user interface layer. Additionally, an Internet Protocol address cannot always be used to learn the identity of a user because there are services that support anonymous Internet activity (e.g., the cloak [Cloak2003] and Anonymizer [Anonymizer2003]). Thus, many services require that a user enter a login ID and password in order that the authenticity of the user can be verified. The MASE tool requires that a user log into the system using a login ID and password before either accessing or modifying protected data. Since the MASE P2P tool is built on top of MASE, the existing login ID and password authentication mechanism was used. Thus, a user must have an account in a peer group before he or she can view MASE data, or view or alter MASE P2P data through the MASE P2P user interface. Also, if a user wishes to copy MASE data to another peer group then the user must have an account in the destination group.

When authentication is required in a system, the use of a session can provide benefits for usability and performance. A session allows state information to be preserved when a user does several transactions, which can have benefits for usability. For instance, this means that a user does not need to log in for every transaction (or request) even if there is a gap of several days in between transactions. For example, the Hotmail e-mail service uses sessions so that a user can access his or her e-mail on different days without having to log in each time. There is a performance benefit because the server can verify the user's identity from a potentially smaller data store than the database of all the users because only a subset of the users will have sessions. Some technologies support sessions automatically, such as HTTP, however the JXTA protocol does not and so session management would have to be implemented by the developer. As a simple alternative to a session, the user could store locally the login ID and password for each peer group in which the user has an account. This way, the user will not be prompted several times to enter a login ID and password when doing one operation, because an operation may depend on data from several peer groups. The performance gains of sessions are not such

an important consideration in the early stages of development of MASE P2P, and the usability benefits are realized by having the user store login IDs and passwords in a local file. Thus, sessions were not implemented in MASE P2P.

Anytime information is sent from one computer to another over a network, there is the potential for someone to eavesdrop on the transmission of information. Data transfer between computers can be encrypted to prevent prying eyes from deciphering the content. JXTA supports encryption using RSA, although the project is recent and still under development [Yeager2003]. For this reason, encryption was not implemented in MASE P2P, but it would be beneficial for this functionality to be included in a more mature version.

Data is owned by a single group, which means that one group is recognized as having responsibility for the original copy of a piece of data (e.g., a file). Although there is no theoretical reason for this constraint, from an implementation perspective it means that locking or synchronization does not need to be implemented. There is no limit on the number of users who can edit information, because depending on the group's membership rules any user could join a group that has ownership of some data. The owner is the one effectively able to decide who can take ownership of, and read, a piece of data. This approach limits the number of copies of a piece of data. Data ownership is used to simplify the synchronization and ownership of data because there is only one original copy of information. This is not to say that a piece of data could not be copied to another location (or machine), but that the copies are identified as being copies and so no synchronization is required between the "originals", which could require manual intervention by a person. Synchronization between originals and copies is addressed in section 4.5 Information Distribution.

One question that arises when you talk about restricting access to information is whether it is possible to restrict un-authorized people from accessing information that has been

accessed by an authorized individual. It is possible to implement a security mechanism that requires a user to enter a login ID and a password every time he or she wishes to access some information, but this is extremely disruptive to the user. Currently, when a file is in an operating system, such as UNIX or Windows, permissions can be set on the file. However, if the file is taken out of the file system and put in a new one then this no longer applies. There is a new technology from Microsoft called Windows Rights Management Services (RMS) that would allow a file to be protected even if the file is no longer part of the file system or in the infrastructure of the company [Wilcox2003]. Although this is an interesting problem, it is outside of the scope of this paper. The MASE P2P tool does not deal with the issue of an authorized individual making available protected information outside of the MASE P2P environment.

4.2 Roles

Roles are used to abstract the functionality and responsibilities of peer groups. One of the essential elements in a P2P environment is the idea of a peer group. A peer group provides a scope for users with similar interests. A peer group is similar to a software development team, but a peer group is a group that is recognized in the P2P network. The network must support the creation of a peer group, and the modification of a peer group's meta information (e.g., description). Users must be able to join and leave a peer group, and be identifiable as being a part of a group. Users must be able to access information about other groups in the network, and be able to communicate with the groups in order to share MASE data (e.g., profiles and tasks). MASE P2P is a secure system that enforces authentication and authorization. Authenticating is used for the verification of a user's identity, and authorization is used to control the access to information. Roles are used to differentiate between users and groups that have different permissions so that a group's IP and the integrity of the IP can be protected.

4.2.1 Information Access

Peer groups have different roles in MASE P2P so that information access can be controlled within the network. Roles are used to define the peer group that has control of a specific MASE object (i.e., MASE-specific data), and the peer groups that are able to copy or take ownership of a MASE object. The peer group that has ownership of a MASE object means that it controls the original instance. Specifically with the workflow objects, the group that has ownership of a task that produces a work product (i.e., variable) also has ownership of the work product because it is responsible for the creation of that product. The owner group can choose which groups are able to make copies (i.e., be a reader) of MASE objects that it owns. It can also relinquish ownership of MASE objects to other groups. MASE P2P does not support the negotiations for a change in ownership because this communication can take place through some other means (e.g., e-mail or telephone). Ownership change is necessary if a module is being outsourced to another development group.

A peer group that can copy a MASE object can be thought of as having read access to the object. Any modifications made to the copy do not affect the original instance. This is similar to the read access used for files in the Windows operating system. For example, if you open up a file twice in a Microsoft Word, the second window will indicate that the copy is read-only, and so if any changes are made to the file then it must be saved under a different name. In MASE P2P, the MASE object's P2P information (e.g., list of reader groups) cannot be edited, and so it is read-only. However, the underlying MASE information can be changed. The copy retains its reference to the original object in case any changes made to the copy need to be integrated with the original. It is somewhat misleading to think of the copy being exact because the MASE data can be altered. If the MASE data is altered then a term such as 'altered copy' would be more appropriate for the P2P designation.

The roles of owner and reader are similar to the roles in open-source projects. Open-source projects often have a small group of core developers that has the final say in what modifications are made to the code that is part of a production release version. In addition to this core group are many developers who can contribute new features and help with bug fixes, but they can only suggest that changes are made to the code. The core developers will review these changes and use their discretion when incorporating them into the code base [Crowston2002]. The configuration management system often mimics this hierarchy of control. For example, a CVS source repository may allow read-access to anyone, but only the core developers are permitted to modify the code. In MASE P2P, a similar idea could be used where the readers suggest updates to the owner of some information.

Users have different roles in MASE P2P. A user can adopt the role of an agent and a P2P administrator within a group. A user can adopt both of these roles at the same time and so they are not mutually exclusive.

If the user has created an account in a MASE peer group then he or she is an agent within that group. An agent can view the profiles of the other agents within the group; however, an agent cannot view the details of a project unless the agent is a member of the team for that project. In other words, a MASE group may be composed of several teams working on different projects. If an agent has access to a MASE object within a peer group using the MASE P2P user interface, then the agent will be able to alter the P2P information for the object (assuming the MASE object is not a copy). This also means that the user is able to view details about the same MASE object through the MASE browser interface. Note that a user has access to some of the same MASE object details using the MASE P2P user interface as with the MASE user interface because the user must be able to identify the objects. However, the MASE P2P user interface is intended for doing P2P operations, and so it does not support the same level of functionality as the MASE user interface for viewing details about MASE objects.

A MASE peer group can specify a P2P administrator password. A user who authenticates him or herself in the group using this password gains access to any MASE object in the peer group and can also alter the P2P data for these objects. This means that a user identified as being a P2P administrator does not need to be an agent in the server peer group. The P2P administrator password is used so that select users can have global access to information within a group in order to do organizational duties to many objects.

4.2.2 Servers and Clients

A peer group can have either server or client responsibilities. Note that more than one peer group could be running on one machine. A server peer group, hereafter referred to as a server, has access to MASE data (e.g., profiles and tasks) because it is assumed to be installed on the same machine where MASE is installed. This means that in addition to the MASE P2P server application being installed, there is a Web server application, EJB application server, and a database. On the other hand, a client peer group, hereafter referred to as a client, only includes the user interface that allows a user to manipulate data in the P2P network. The client is a lightweight entity compared to a server because it only needs to run the MASE P2P client software.

Servers and clients store information about other groups that includes the details (i.e., routing information) needed in order to communicate with other groups. It is useful to be able to find this information about a group even if it is offline and so peer groups can retrieve these details indirectly from trusted peer groups, which are essentially providing a centralized service (see Figure 4-1). The use of a centralized service is not uncommon in a P2P network. Napster was a brokered P2P service in that all searches were done on a central server, but then the files were transferred directly between peers. Even in the completely decentralized Gnutella network, super peers store information about nodes beneath them so that searches can be more efficient. The benefits from this approach are that the network traffic was reduced due to fewer packets being sent on the network, and there was less dependency upon slow machines for providing responses. In fact,

machines that are expected to be always online are often invaluable in a P2P network. For example, a server could provide a certification service for peer groups.

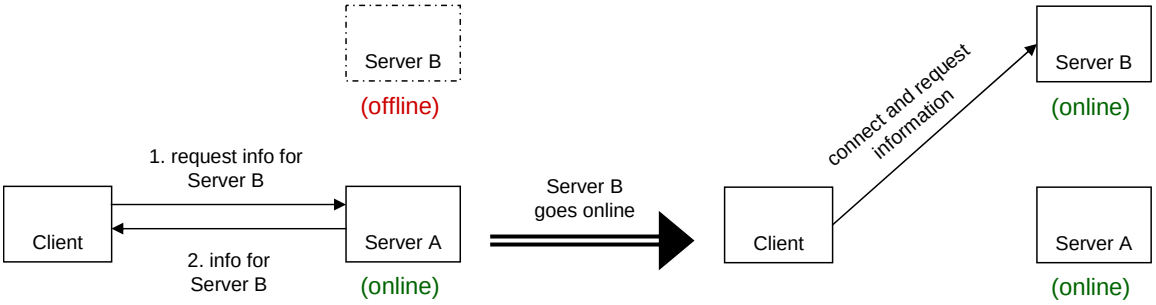

Figure 4-1. Known and trusted groups can provide the information required for how to connect to other groups. The client request the information from Server A on how to connect to Server B, which is offline. The client can then connect to Server B when it goes online because the client has the information needed to communicate with Server B.

4.3 Information of Interest

The question of what information is available in an application and can be moved from one site to another is dependent upon the domain and requirements of the application. For MASE, there were three areas that were thought to be worth considering as being relevant to intellectual property.

1. *Resource pool*. The resource pool contains information on the people who have accounts in a group (i.e., their profiles). A profile includes contact information and the person's skills. Someone who has a profile in a group is referred to as being an agent of the group.
2. *Workflow*. The workflow contains the breakdown of projects into the tasks required to complete the development.
3. *Experience base*. The experience base contains templates of workflows for projects. The templates are like skeletons that can be applied to similar projects.

For MASE P2P, information from the resource pool and workflow were considered. It would be beneficial for developers if MASE P2P handled the information in the

experience base, but for the purposes of this research it is not necessary because the experience base contains information that is very similar in structure and role (i.e., managing the tasks in a project) to the information in the workflow.

4.3.1 Units of Information

MASE is written in Java, an object-oriented language, and so objects can be thought of as logical packages, or units, of information. The following are the units of information found within the resource pool and workflow: agents, projects, processes, and variables. For simplicity, these four units of information are referred to as MASE objects.

Agents represent team members and are part of the resource pool. Information about agents is made available in the P2P context so that people can search for agents (e.g., developers and managers) in other groups. A user can have a different agent profile in each group that it has created an account. An agent can be searched based on contact information or skills, which may be useful if a developer with specific skills is needed. For example, a group may need to find a developer with J2ME experience to develop an application for wireless devices.

A workflow is broken down into projects, processes, and variables (Figure 4-2). A project includes a project name, description, and the manager responsible for the project. A process represents a task, and is dependent upon a project. A process includes a type (i.e., iteration, user story, or task), a name, description, the project it is part of, and where it is located in the workflow hierarchy. MASE follows XP practices, and so these types are described in that methodology [Beck2000]. For example, a project to develop a Web application may consist of the broad tasks of creating the HTML interface and the business logic. A process can be divided as many times as is necessary in order for the developers to adequately handle the complexity of fulfilling the requirements of the project. A variable represents a work product for a task, and is identified as being a certain type, such as source code, or a text file. For example, one of the work products of a programming task would be source code. In MASE, a process produces only one work

product, although that work product can be the input for several processes. At the time of development of MASE P2P, variables were not part of the current version of MASE because they were not absolutely necessary, and so variables were not included in MASE P2P. In the future, if variables are re-introduced into MASE, they could be included into the workflow support in MASE P2P.

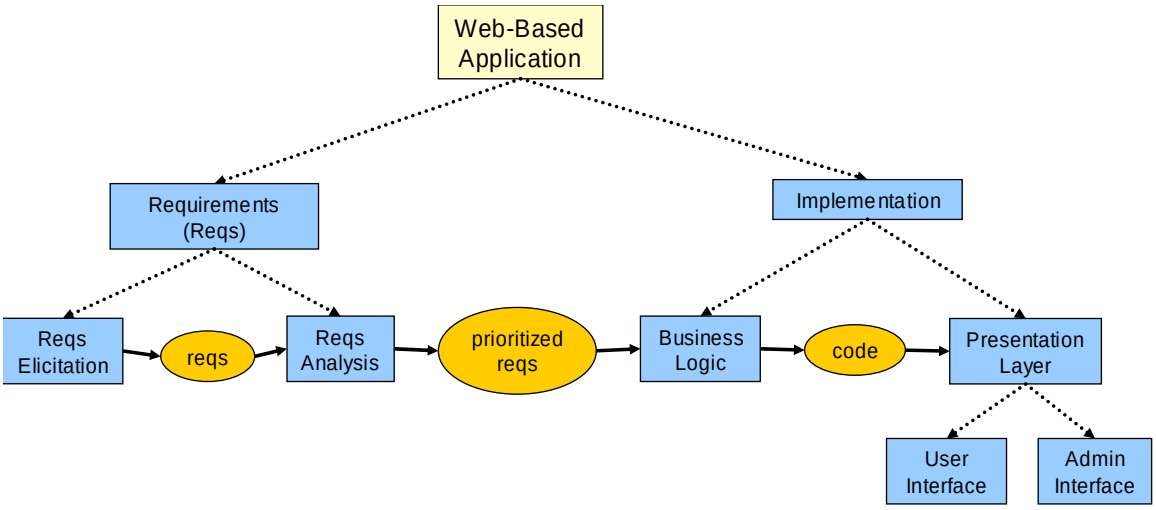

Figure 4-2. An example of the workflow breakdown for the tasks of producing requirements and implementation in a Web-based application. Processes (i.e., tasks) are represented by the squares, and variables (i.e., work products) by the circles.

To summarize, the following MASE objects are of interest in the MASE P2P because of their relevance to the IP of developers and organizations.

- Agents from the resource pool
- Projects from the workflow
- Processes from the workflow

4.4 Information Organization

Deciding how to organize the information in an application can have an effect on the complexity of the software (which will affect the difficulty of implementation), and the

performance of the application. Information can be more easily understood when it is considered within a context, or in other words grouped together with related information. An object's type and its relationships with other objects help in the understanding of the object and its data.

From an implementation viewpoint, it is better to group tightly coupled items at one site so as to avoid the overhead of coordination and synchronization across multiple sites [Mockus2001b]. Work has been done to identify ways of automatically clustering objects that are highly dependent upon one another [Kim1998]. The purpose for clustering objects that are dependent upon one another is that the network load and load at one machine can be reduced. Objects can be clustered manually or automatically, but manual clustering usually requires an experienced developer to cluster the objects. The cluster size will also affect the efficiency. Grouping more information together means that synchronization may take longer and unnecessary information may be transferred when data is moved around; however, it reduces the amount of complexity in grouping the information. The optimal solution for clustering is to have information broken up into units that are as independent (i.e., loosely coupled) and cohesive as possible.

In MASE P2P, the decision for how to organize, or cluster, MASE objects (i.e., agents, projects, or processes), resides with the user. MASE objects are grouped and moved together based on the user's need, and not with the goal of minimizing overhead.

4.5 Information Distribution

Information distribution refers to the placement of data in the P2P network. There are several factors worth considering in MASE P2P.

1. *Does the system support copying data within the network?* Duplicating information is useful because redundancy increases the robustness of the network. Also, information can be brought closer to a user, which may reduce the delays

for the user to access the information. Ideally, the information could be copied and placed on the user's local machine.

2. *Are copies considered equals with the original?* In other words, can a copy be modified and still retain a reference to the ancestor, or other originals? This question has a bearing on data synchronization.
3. *Who is able to make copies?* Permissions are used in operating systems to define who is able to read, modify, and execute files. The same idea can be applied to data in a distributed environment.

These questions are addressed and discussed in detail in the following sections: data duplication in section 4.5.1 Data Duplication, data synchronization in sections 4.5.2 Data Synchronization and 4.5.3 Data Synchronization Initiation, and permissions in section 4.2.1 Information Access.

4.5.1 Data Duplication

There are several benefits when MASE objects can be duplicated, or copied, from one machine to another. It is important to note that duplication in this section means the system-supported copying of data. After all, given enough time, a user could manually copy any information that he or she is able to view. (The issue of who is able to view information is addressed in the section 4.2.1 Information Access.) Understandably, this is an inefficient means of duplicating information, and it could be nearly impossible to do if the data is large and/or complex, such as an intricate graph structure. The benefits of supporting data duplication are summarized here, and then addressed in more detail after the summary.

1. Reduced request time
2. Increased redundancy
3. New knowledge can be gained

Firstly, if data can be duplicated then the user can benefit from having to wait less time when accessing the information. For example, Akamai is a company that offers a service

called EdgeComputing that uses data duplication to reduce demands on IT (Information Technology) infrastructures by storing copies on many servers around the world so that users are able to have faster access to information [Akamai2003]. There are several factors worth considering for where to put a duplicate copy or copies because the following factors can affect the length of time required to download information from a machine having a server role: the speed of the server, the number of requests the computer has to deal with (i.e., the load), the amount of available bandwidth on the line on which the machine is connected to the Internet (i.e., amount of congestion), and the distance the server is from the user.

Secondly, if the machine on which the information resides is not available, or online, when the user makes a request then the user cannot access the information. For example, it can be extremely frustrating when trying to access a work-in-progress paper from a network server when the server is unavailable. Even popular services that are funded by large corporations are sometimes inconveniently unavailable. For example, the Microsoft Hotmail mail service is the world's largest free e-mail server, with over 50 million users, and yet the service is occasionally unavailable ([McCullagh2000], [Delio2001]). When there is redundancy in a system, there is a greater possibility that information will always be available.

Thirdly, being able to duplicate data provides an efficient means for supporting new knowledge. A copy can be tailored to a different context or situation. The workflow breakdown structure for a project can be used as a template for another project. For example, a team could make use of a workflow for a Web-based application from another team as a guide for the development of their own Web-based application.

4.5.2 Data Synchronization

Data synchronization involves merging data from two different entities. The two entities are usually related in some way. For example, they may have similarities because one of the entities is an altered copy of the other, or they may both be copies from the same

ancestor but were modified differently. Synchronization would allow copies of an original to be updated to the most recent version of an original. CVS is a version management system that supports data synchronization so that several developers who are working on the same source code can combine their changes into one version. A user can decide how to deal with any conflicts between two files after seeing the differences between the files. A manual process is required because changes from an older version of a file may need to be incorporated with newer changes. If data synchronization is not used to deal with multiple copies of a file then a locking mechanism would have to be used to prevent users from altering more than one instance of data at one time. Locking has the disadvantage of preventing users from being able to work on one file at the same time.

The syntax of the information in two entities must be compatible and comparable for synchronization to be possible. For example, two text files can be compared if the text is encoded using the same conventions (e.g., UTF-8). As well, it must be possible to identify that the two instances came from the same ancestor. Meta data can be used to make this identification possible, for example, using a unique identifier to identify that two objects have the same ancestor.

Of course, if two entities are identical then they do not need to be synchronized. Two methods that can be used to determine whether two entities are identical are with the use of a checksum or timestamp, or a combination of the two. A checksum is a value that is computed based on an entity’s contents--there is a high probability this value will change if the entity changes. A timestamp is a chronological stamp for the last time an entity was changed. The timestamp approach should use a centralized time server in order that all machines are synchronized with the same clock. For simplicity in the MASE P2P implementation, only the timestamps of objects would be compared when determining whether objects need to be synchronized.

Whether automatic data synchronization is possible is dependent upon how many copies exist and whether they can be edited, or if they are treated as read-only copies. The different scenarios are outlined in Table 4-1.

Table 4-1. Types of synchronization possible for distributed information.

Number of Editable Instances	Duplication	Synchronization
One	No	n/a
One	Yes	Automatic update is possible
More than one	Yes	Manual update is required if no locking mechanism is used

If there is only one instance of an editable original piece of data, then it is possible for copies to be automatically synchronized. That being said, in a P2P environment, there are no explicit guarantees that a machine will be online, and so a mechanism would be required in order to ensure the copies are made aware of updates. For example, a central machine, or several machines, could be used as a store of changes to data in the network, and when a machine goes online it could update its copies with the changes.

A locking mechanism could be used if there is more than one instance of a piece of data, but only one instance can be edited at one time. Some type of flag could be used to indicate whether an instance is being edited. Once an instance has been edited, the changes could be propagated to the other copies. If there is more than one editable instance and no locking mechanism is used then a user would have to manually confirm synchronization changes, because changes could be done simultaneously on different instances.

The possible scenarios for data synchronization were examined, and the implications for the amount of effort required for the implementation considered. For simplicity in the initial implementation, it was decided that only one instance of a piece of data would be recognized as being editable in the network. This means that no support would need to be implemented for the user to do a manual synchronization of two instances of an entity

bean. Furthermore, as a consequence, in a later version of the MASE P2P system, automatic synchronization would then be possible.

4.5.3 Data Synchronization Initiation

Data synchronization is a process involving several steps: 1) there is an initiation or request that two pieces of data are compared, 2) a comparison is done, 3) decisions are made as to what changes are to be made to either copy, and 4) the changes are made to the respective copies.

The first and last steps are interesting from the MASE P2P perspective, because they are tightly related to the organization (i.e., distributed nature) of objects in the P2P environment. The second step can be handled by third-party code, and the third step simply requires the user making decisions. The first step will be looked at in more detail, because it involves identifying and finding appropriate objects in a distributed environment. The process begins with an object that is identified as being a copy, or has associated copies, changing. The change can be represented by an advertisement being generated that details the object that was updated and the changes that took place. There are two methods that can be used to help initiate synchronization: push and pull. A push mechanism is known as an unsolicited method of advertising because the peer that wishes to propagate the advertisement simply does so even though the recipients did not solicit the information. The pull mechanism is a solicited form of advertisement retrieval because the peers will ask for updates when they choose to do so.

Consider the scenario where peer group X is in charge of (i.e., is the owner) the original version of an object, and peer group Y has a copy of the object. There are two different ways that group X could push an advertisement regarding a change to the aforementioned object: a) peer X can broadcast the change to everybody, or b) peer X can multicast the change to only those groups that have a copy of the information. Option A is a waste of bandwidth, while option b requires the developer to keep track of who has copied each object. Option b is the preferred choice when it is a priority to minimize network traffic.

Additionally with option b, a push can be done at a later date if one of the peer groups is not online to receive news of the update.

Alternatively, a pull mechanism could be used where a peer group will poll to see whether an update has been made on the original copy of a piece of data. It would be detrimental if a periodic poll was done since many unnecessary polls could be done if the objects do not change often. Additionally, a peer group may have many copies of objects, which could lead to machines getting overloaded with a multitude of requests. The most efficient pull approach would be if a poll is made only for an object that the user is interested in, for example if a user is going to view, or perhaps update. However, this approach may cause noticeable delays to the user because the check would have to be done when the user requests an operation on an object.

Using a hybrid push and pull approach minimizes the load on the server. Whenever a server makes a copy of an object, the copy will have the same URN (i.e., location and time independent unique identifier) as the original so that the two can be identified as being related. The peer group that has the original object will have a table that is used to keep track of the peer groups that have copies of the object. If an original is altered, then an attempt would be made to contact the peer groups that have the copies to inform them of the changes. This is the push part.

For the first step in the approach, an attempt is made to push information from the peer group that has the original object (i.e., owner group) (see Figure 4-3). This step would need to begin with the owner peer group sending a request to a peer group with a copy (i.e., copy group) that it wishes to send an updated object. Both of the peer groups would need to be authenticated at this time so that both parties are confident that each peer group is who it says it is. At this point, the following cases are possible:

1. *update successful*. The copy group tells the owner group to go ahead and send the updated data. The update is sent, and the copy group acknowledges that the update was successful.
2. *copy does not exist*. If the copy group responds that it no longer has a copy of that piece of data then the reference in the owner group that links the data to the copy peer group is deleted.
3. *copy peer group is offline*. If the copy group is offline then the update information is stored in the owner group, or within a trusted third party group. A third party group would have to be used if the owner group is not always online because the information must be accessible all the time. The third-party group would have to be trusted because although the actual object data is not being stored on a third-party machine peer groups are dependent upon having accurate information about objects that have been updated. The update information would include the copy group id, data id, modification date and time, and owner group id if the information is stored in a third party group. This information would be replaced if a subsequent update is performed on the same object, or deleted if a certain period of time elapses.
4. *update unsuccessful*. If an attempt it made to the update the data on the copy group, but the update is not successful then the update information is stored as described in case 3.

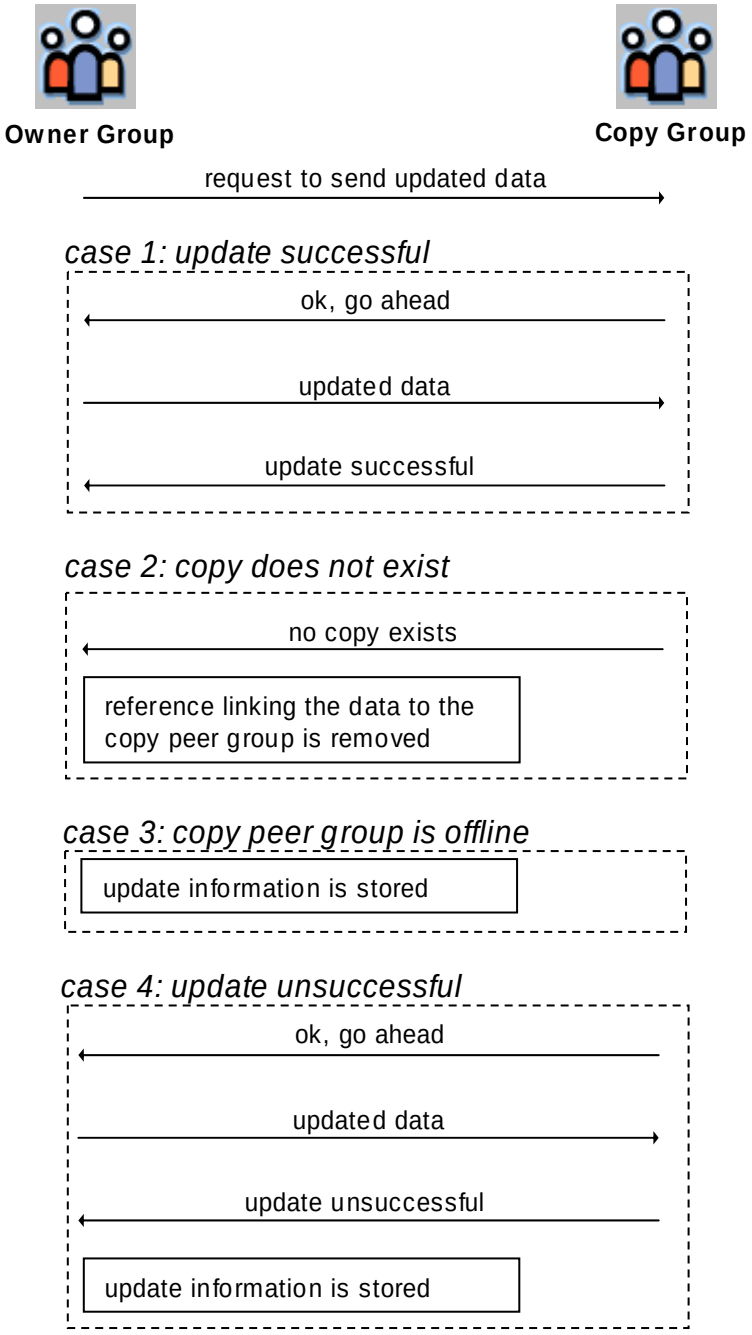

Figure 4-3. An update is pushed from the owner group to the copy groups. The update information includes the id of the peer group whose copy could not be updated, and the id of the data object that was updated.

The push step is followed by the pull (see Figure 4-4). When a peer group that was offline goes online, it should check (or pull) for whether any of its copies have been updated. A pull could be done for all copies when the peer group connects to the network, but this could take awhile if the peer group has many copies. Or, the pull could be done only when a specific copy is viewed by the user. The following cases may result from this request:

1. *no updates exist*. If no updates exist for the data then nothing further needs to be done.
2. *updates exist and successful update*. If there are updates for the data, and the update is successful then the reference in the owner group or third party group can be removed for the copy group that retrieved the updated data. If no more updates need to be done then the update information can be removed.
3. *updates exist and unsuccessful update*. If there are updates for the data, but the update is not successful then nothing further is done so that the copy group can attempt to obtain an update at a later date.

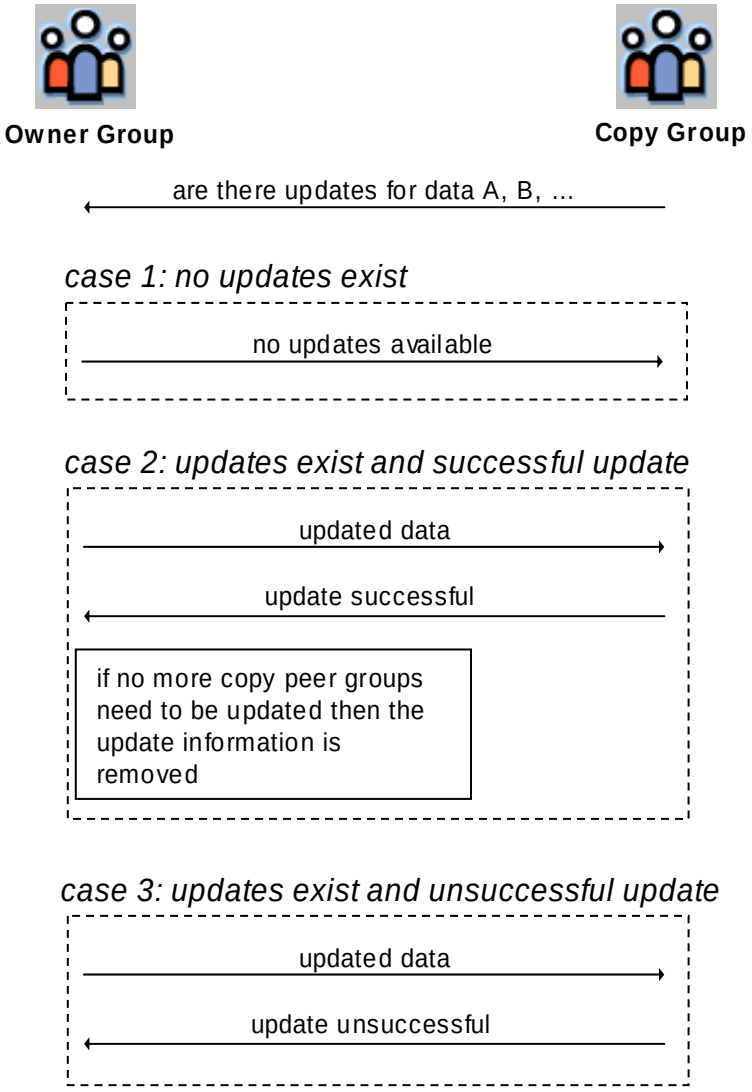

Figure 4-4. An update is pulled by a copy group from the owner group.

A mechanism for implementing data synchronization when only one peer group can be an owner was described in this section. Because of the amount of effort that would be required to implement the data synchronization between an original and the copies of an object, and the time constraints for this research, this feature was not implemented in MASE P2P. If more than one peer group could have original (i.e., editable) copies of an object, then a similar mechanism could be used to notify the owner peer groups, but some

negotiation mechanism would also be required in the event that two originals of an object are altered at the same time.

4.6 Data Transfer

When people or applications communicate with another there must be an agreement on the rules of communication. These rules should include both the syntax and semantics of the language. The syntax includes the rules for defining how pieces of data look and fit together, and the semantics refers to the meaning. For effective communication to take place, the parties must agree upon and understand the syntax. Once the information is recognized, the meaning (or semantics) of the message can be understood.

XML is a common method of structuring information for data exchange [XML2003]. XML is a flexible, extensible language that is used to structure information using markup tags. The information is represented as ASCII text and so it is human-readable with any basic text editor, and can be parsed using any programming language. Binary information can also be sent using XML. There are several tools and APIs in most of the popular programming languages that support the parsing and generation of XML documents (e.g., WSAD [WSAD2003] and XML Spy [XMLSpy2003]). XML is used in JXTA for advertisements and messages, and is used for all configuration files in MASE P2P. XML could even be used to represent MASE objects so that non-Java programs could interpret the data, but for the purposes of this paper it is assumed that only the Java implementation of MASE is used. XML is an extremely popular and useful technology that is used in MASE P2P.

Data can be transferred in many ways over a network. Obviously, there are physical considerations at the physical layer; however, there are also decisions at the application layer, which are of interest in this paper. One consideration is the port used to send and receive the data. There are some ports that are reserved for certain protocols (e.g., port 80 for HTTP, and port 21 for FTP [File Transfer Protocol]). However, theoretically there are

an unlimited number of ports available on a machine. Although, an administrator will usually limit access to most ports in order to prevent un-authorized access. A firewall is a machine that is used as a proxy between machines outside of a network and those machines inside of a network in order to prevent unauthorized access to machines within a local network. They can also be used to monitor incoming and outgoing traffic. Some technologies, such as Java RMI (Remote Method Invocation), will make use of an uncommon port in order to take advantage of a quiet port, but can automatically make use of a common port such as port 80 in case the other port is not available [RMI2003]. Port 80 is usually open because it was normally intended for Web HTTP traffic, and so applications can usually tunnel through port 80. Although this means that most traffic can pass through port 80, security can still be enforced because traffic can be monitored and individual packets filtered. Thus, port 80 can become a bottleneck.

The JXTA technology allows applications to connect to the network via a proxy server, and through a firewall. The port used can be automatically, or manually, selected by the user, which provides a great deal of flexibility for the MASE P2P applications to connect and transfer data in most network configurations.

4.7 Searching

Search capabilities are needed for both identifying peer groups and MASE content. A P2P environment can be quite dynamic in that users can log in to a network and immediately make available a large amount of new information in the network. Since users need to know about the new information, meta data can be published that allows users to find out what information is available and where to obtain it.

For searching to be user-friendly, a user should be able to make a search query based on several criteria. For example, a project may require a developer with a specific skill that cannot be fulfilled by a member in the current group. Thus, a search should be possible based on data such as name, skill, and group. Additionally, for user-friendliness, a search

query should be flexible in that a query can be done with very little information in case the user wants to look through a maximum number of results. A search should allow the user to search using several fields of data. Many search engines on the Internet give the user the choice of doing a basic search where the search criteria just needs to be entered into one text field, or doing an advanced search where the search criteria can be divided according to the fields of interest. A simple solution for the developer is to have a single text-box that can be used to enter field-value pairs for the query. It is not trivial to create a detailed search mechanism that allows for the user to choose Boolean operators for information (i.e., AND, OR, and NOT), and so an alternative is to use regular expressions for the query. Although this approach requires the user to know the syntax for regular expressions to do advanced searches, simple searches are still possible without any knowledge of regular expressions (e.g., putting Java will search all fields for the text ‘Java’).

There is the potential for a great deal of information to be returned from a search, which means features are needed that allow the user to efficiently navigating through large amounts of information. For example, a user can more easily find a row in a table if the rows can be sorted by column. Also, labels for some objects are differentiated by characteristic such as font style, colour, or logo so that the user can quickly inspect many objects to identify the ones of interest.

5. MASE P2P IMPLEMENTATION

The purpose of this section is to discuss some issues that arose when implementing MASE P2P, and to further explain the tool's functionality. In this section, classes are often represented in logical packages called modules. A module is an abstract conceptual container rather than a physical package. A module includes classes that have some unifying responsibility or function. The modules were not designed beforehand; they evolved during the implementation of the tool. In general, classes that were related (i.e., part of one module) were initially implemented around the same time because they were related and dependent upon one another. Design patterns were used in order to reduce the complexity and increase the maintainability of the code.

The author, who was also the developer, made a conscious effort to reduce the coupling between modules. In any application, the presentation and business logic modules are often implemented with the goal of having a loose coupling so that code in the two modules are flexible and can be reused [Fowler2003]. One requirement for loose coupling is for the two components to have well-defined interfaces. Minimizing the coupling between two modules results in having to do fewer changes to the implementation in one module if changes are made to the other. Reduced coupling between modules was often a by-product when an attempt was made to use inheritance in order to re-use code between modules. Abstract classes were implemented in order that code could be re-used between modules, and interfaces were used in order to enforce a well-defined interface between classes in different modules. Using inheritance proved to be invaluable from a maintenance perspective because the code was easier to understand and modify.

5.1 Programming Language

Java was used as the programming language for a number of reasons:

- *Interoperability*. Java was used for the implementation of the MASE system and so some of the same applications could be used to deploy the MASE P2P system. More importantly, it is simplest if code written in the same language interacts together rather than code written in different languages. The JXTA P2P API (Application Programming Interface) that was used for the P2P functionality was also written in Java.
- *Object-oriented*. Java is an object-oriented language, which means it is extensible, there is a natural abstraction for compartmentalizing functionality with well-defined interfaces, and code can be easily reused with the use of inheritance. As well, the object-oriented abstraction allows for a natural mapping of real-world objects to Java classes
- *Resources*. There are many resources (e.g., tutorials, APIs) available for Java developers at http://java.sun.com. Programs written in Java can make use of the J2EE (Java 2 Platform Enterprise Edition) technology, which allows for remote method invocations and mapping objects to a relational database for persistent storage using Enterprise Java Beans (EJBs) (see Appendix B: Enterprise Java Beans).
- *Familiarity*. The author, who is also the developer, is most familiar with the Java language. One IDE (Integrated Development Environment) can be used for developing the MASE P2P source code and viewing the MASE source code.
- *Portability*. Programs written in Java can be deployed on several platforms (Windows, Macintosh, Solaris SPARC 32-bit, Solaris SPARC 64-bit, Solaris x86, Linux [JVM2003]) without having to do modifications to the source code.

5.2 Incorporation of P2P Functionality with MASE

The MASE P2P implementation was deployed in a separate module from the MASE implementation. MASE is packaged within a single EAR (Enterprise Archive) file, and the MASE P2P code was also deployed in its own EAR file. There are two advantages with keeping these components separate. First, the MASE code is not dependent upon MASE P2P code. There is a dependency from the MASE P2P module to the MASE module, but not the other way around. Thus, if there is a bug in the MASE P2P system then it will not disrupt the operation of MASE, which contains the tool's core functionality. For example, an application server will sometimes need to be restarted if an exception is thrown and not caught (i.e., there is a case that is not handled) from code from a deployed EAR file. The second advantage is that the MASE P2P does not need to be integrated into the MASE code. For example, the author did not need to understand and then alter any existing UI code. As well, since MASE is going through continual changes, the majority of which affect the UI, the MASE P2P UI code would not need to be changed whenever the MASE UI code went through a change.

A new entity bean was created that contains the P2P meta data, as well as a reference to the MASE entity bean that it refers to. Enclosing the P2P information in its own entity helps to maintain the cohesiveness of the entity beans because not all MASE objects will need to have P2P information associated with them.

There are two main disadvantages with MASE and MASE P2P being separate modules, both of which affect the end user. Installation of the P2P functionality requires downloading and deploying a separate package from the core MASE package, which requires more work by the user. The second disadvantage is that because the two components are deployed separately in the application server, RMI calls are required between the two components. Remote calls have more overhead than local calls (even when they are called from programs running in the same JVM [Java Virtual Machine]),

because of the JNDI (Java Naming Directory Interface) lookups, and subsequent network communication. It was felt that the advantages outweighed the disadvantages.

5.3 Architecture

An application architecture was used for MASE P2P. Changing the architecture of a system can take a significant amount of effort. The final architecture came about as an emerging design because other architectures were attempted first. These architectures are discussed in detail in section 6.1 Architecture of chapter 6 Lessons Learned.

When two different pieces of software having different functionalities interact in a distributed environment, there is the chance that extensive changes may be required to both modules if the implementation responsible for the communication is tightly coupled to the rest of the software (i.e., the so-called business logic). The client and server software packages are essentially modules with different purposes that communicate with one another across the JXTA network. The extension of abstract classes was used in order to re-use code in the communication module (Figure 5-1). The interfaces were simple in that although several different object types could be transmitted between peer groups, a generic interface was used for all object types. Thus, new object types could be sent without having to change the methods responsible for communication. On the receiving end, an abstract class called *MessageListener*, which served the purpose of a controller, understood the format of the incoming messages. The client and server modules extended *MessageListener* in order to link to the back-end.

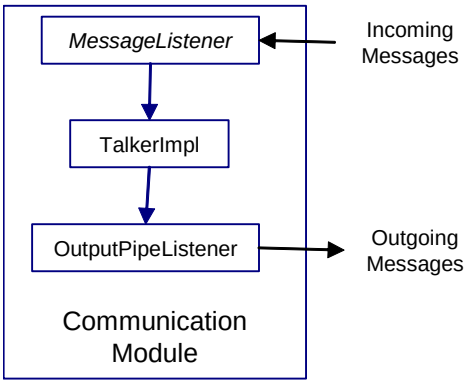

Figure 5-1. The communication module contains objects with well-defined interfaces in order that the coupling between the communication module and business module was minimized. The abstract *MessageListener* class is extended in the client and server modules. The *TalkerImpl* class is independent from code responsible for the client and server business logic and so it is used in both the client and server modules. The communication module is responsible for sending and receiving messages to and from peer groups, respectively.

A fair amount of the JXTA code responsible for the creation and initialization of a peer group was re-used between the client and server modules. The peer group module contains an abstract *DeveloperGroup* class that is responsible for the creation and initialization of the peer group. It makes use of an abstract *Preferences* class, which is extended in the client and server modules. The syntax of the client and server preference files is different because they each contain some unique information. A generic *AdvertisementListener* was used to listen for peer group advertisements that are published in the JXTA network.

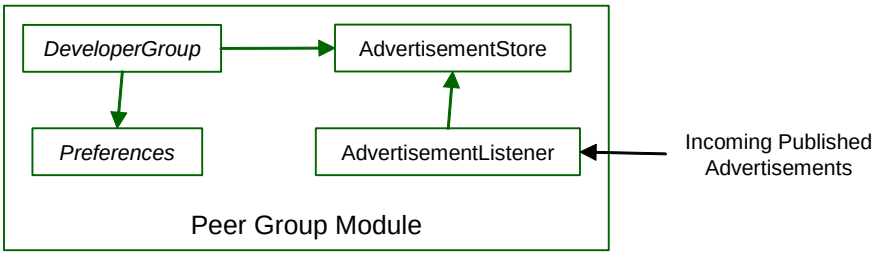

Figure 5-2. The peer group module is responsible for the creation and initialization of the JXTA peer group. *DeveloperGroup* and *Preferences* are both abstract classes that are extended in the client and server modules. The peer group module contains group preferences (e.g., user name and password in the JXTA network), and a store of advertisements that are published by other peer groups.

There was an effort to minimize coupling between the implementation for the user interface and the rest of the code, as described in the Model-View-Controller design pattern [Fowler2003]. For example, in the client, the View was represented by the *MainGUI* class, which was responsible for displaying the GUI objects. The Controller was represented by inner classes within *MainGUI* that handled events and the *Client* class that provided an interface between the presentation and business code. The use of classes to decouple the presentation from business logic reduced having to make changes to both modules if changes were made to one module. The architecture of the client module is shown in Figure 5-3.

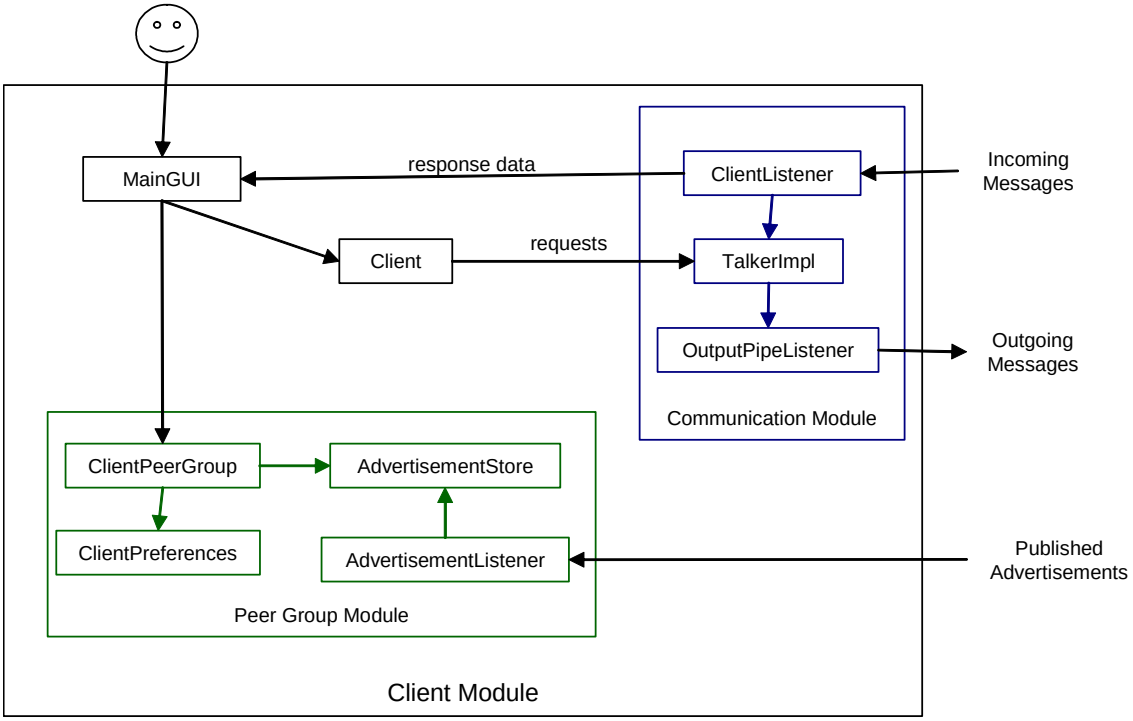

Figure 5-3. The user interacts with the client system through the *MainGUI*. The peer group and communication modules are created by the *MainGUI* when the user launches the application.

The session façade and value object design patterns were used in the implementation of some of the classes in the server module (Figure 5-4). The session façade pattern is frequently used with EJBs to separate the persistent data storage implementation from the rest of the software. The value object pattern is a natural extension to the session façade

design pattern because it allows for the complete separation of non-EJB code from the entity beans, which provide the mapping of the objects to the database.

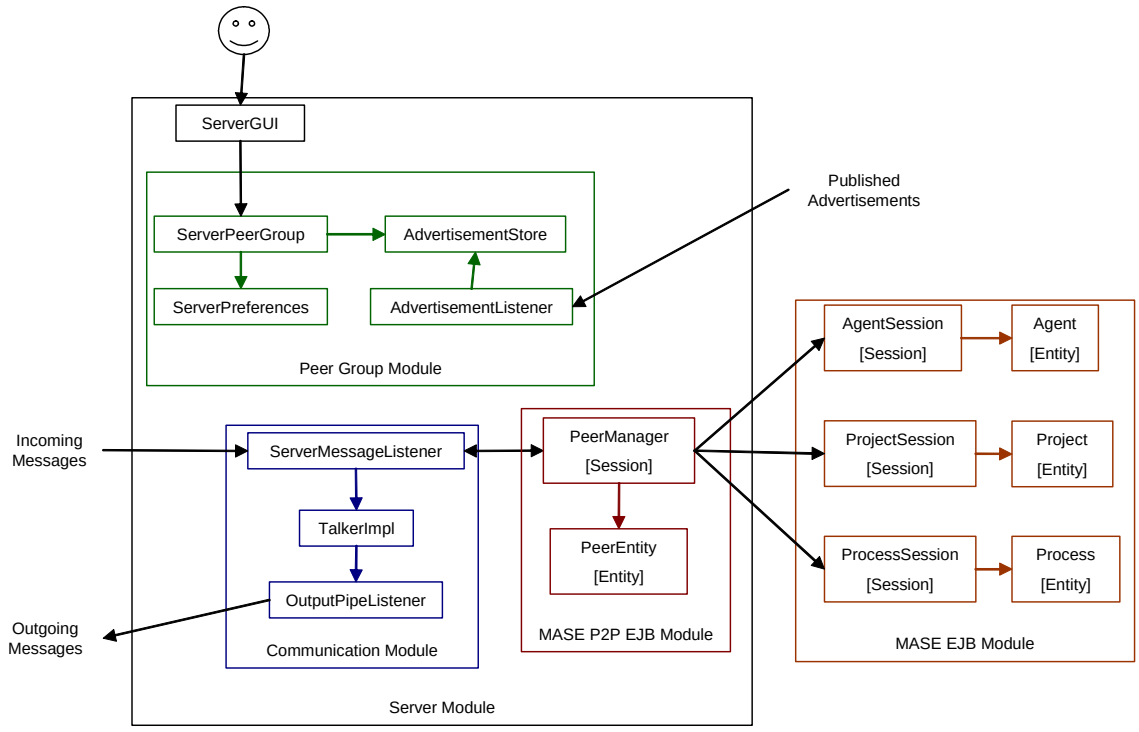

Figure 5-4. The user interacts with the server module through *ServerGUI*. The peer group and communication modules are created by *ServerGUI* when the user launches the application.

5.4 JXTA

The JXTA (pronounced “juxta”) framework was used for the P2P functionality, because implementing a new P2P infrastructure would be a research project in itself. JXTA was spearheaded by Sun Microsystems, but now it is its own open-source project (http://www.jxta.org). The name stands for juxtapose, because it was intended that P2P applications using the JXTA protocols could exist alongside the existing, and more common, client-server architecture.

JXTA is a general framework in that the six protocols that make up the specification are independent of programming language, platform, and network transport protocol [Gong2001]. For example, an application that makes use of the JXTA network could be

written in Java or C#, and could run over TCP (Transmission Control Protocol) or HTTP. It provides a common network architecture layer for applications, much like the TCP/IP network protocol. There is no dependence upon any kind of master directory or channel at which devices must register to make services available, or to be able to search for other services. This is different from several other service discovery protocols, such as Salutation [Salutation2003], and Jini [Jini2003]. The protocols are ideal for P2P networks, which are inherently heterogeneous and flexible networks, because the protocols were designed to be lightweight, which means that small portable devices (e.g., cell phone, Palm device) could interact in a network with a small footprint. The first version of JXTA was implemented using Java, but it is being implemented in several other languages, including J2ME (Java Micro Edition) C/C++, Perl, Python, and Objective C [JXTA2003].

JXTA makes use of two fundamental constructs, peers and peer groups. A peer group is a collection of peers that have some kind of unifying interest, and it can support a secure domain for the exchange of private information. When a peer group is created in JXTA, it is automatically part of the JXTA world peer group, as well as a unique peer group. Each peer must be part of a peer group. All peers in the world peer group understand the JXTA protocols and so are able to communicate with one another.

The JXTA implementation is found within the peer group and communication modules. A significant amount of the time spent implementing MASE P2P went into creating an API of sorts for working with the core JXTA API. When development began on MASE P2P, the first version of JXTA had only been around for one year (i.e., since approximately May 2001), which meant that most of the books available only included simple and sometimes incomplete examples. Furthermore, there was little help available because there was no mailing list for developers to post questions related to the technology. For example, only after a difficult period of trial and error was the author able to figure out that a JXTA peer group cannot be created within another peer group,

which was initially seen as being an essential part of MASE P2P. Only through membership can a peer be abstractly considered to be part of multiple groups. Users would have to create accounts in all of the different peer groups they wished to join, which amounts to the same result in the end, but it requires more effort on the part of the user, and so it is a less elegant approach. For example, if one group wanted to join with another group then they would have to physically join all of the data. You can read more about JXTA in Appendix C: JXTA.

5.4.1 Limitations

The main limitation with JXTA is that it is a relatively new technology. The first stable release was made available in May 2001, and version 2.0 just recently. As with any new technology, the JXTA technology is continually evolving. Many general methods were implemented during the development of MASE P2P in order to supplement the JXTA API; they were general in that they could be used in any application. From reading over the release notes for version 2.0, some of this functionality has been incorporated into the JXTA API, which means that the MASE P2P code could likely be refactored and cleaned up.

Another limitation with the JXTA technology is that only sub-group can be created below the world group. For MASE P2P, a minimum of two levels are needed below the world group because there is a MASE group (i.e., all the groups that are sharing MASE information), and then individual groups below the MASE group. This limitation was circumvented by including a uniform prefix for the names of the peer groups that were part of the MASE group, which amounts to the abstract representation of a group. This is not as elegant or simple as having an actual sub-group, because the hierarchy cannot be used to make use of potentially inheritable traits like permissions. The abstract approach would also become increasingly complex if users desire additional levels of sub-groups.

5.5 User Interface

The user interface allows the user to search for groups, access the information within a group, and alter the information. It is cumbersome and awkward to use an application if the user interface is unintuitive and unnecessarily complex, and so an effort was made to make the graphical user interfaces (GUIs) simple and user-friendly. The GUIs had the following characteristics.

- *Flexibility*. The main window can be resized to fit the user's entire display. Also, sections of the window can be resized if the user wishes to enlarge an area of the window, such as the table of peer groups. Some information can be filtered in tables in case there are many rows in a table. For example, in the peer group table, the user can choose to only see client groups, server groups, or both. As well, the user can choose to jump to the row that applies to one of the groups for which he or she has an account. The user can click on any of the column headers in the agent or peer group tables to sort the rows ascending or descending by the information in that column. Furthermore, the user has the choice to do operations on several MASE objects at one time, which saves the user from having to repeatedly do a mundane action.
- *Consistency*. Several of the menu items and menus are consistent with Windows applications. For example, there is a *File* menu with an *Exit* menu item that is used to close the application. As well, the application will use a look-and-feel that is consistent with the operating system.
- *Intuitiveness*. Frequently used operations are accessed directly in the main window, and less commonly used operations are accessed via the menus. Also, the most important objects in a window are situated near the top.
- *Helpful feedback*. Feedback to the user is important so that the user knows whether an operation has been successful or not. Note that the absence of feedback can also be an important method of conveying a message. For example, in the UNIX operation system, the "no news is good news" approach is a non-intrusive feedback mechanism. For the MASE P2P GUIs, short messages are

shown in the bottom right-hand corner for non-critical information that may be of interest to the user. It serves the same purpose as the status line in a Web browser. A modal dialog window is used to inform the user of more important feedback, such as an error message if an operation could not be performed. If an operation fails on several objects then the user can view why the operation failed on each affected object.

- *Aesthetically pleasing*. Simple colours were used to make the interface aesthetically pleasing. The look-and-feel of the window is handled automatically by the JVM, which will use the look-and-feel most appropriate for the operating system. For example, if a user is running the application on a Windows machine, then the look of the interface will resemble the Windows look. The Windows XP colours are simple, with the majority being white, a light beige colour near the outside of the windows, and blues. A screenshot of the MASE P2P client GUI is shown in Figure 5-5.

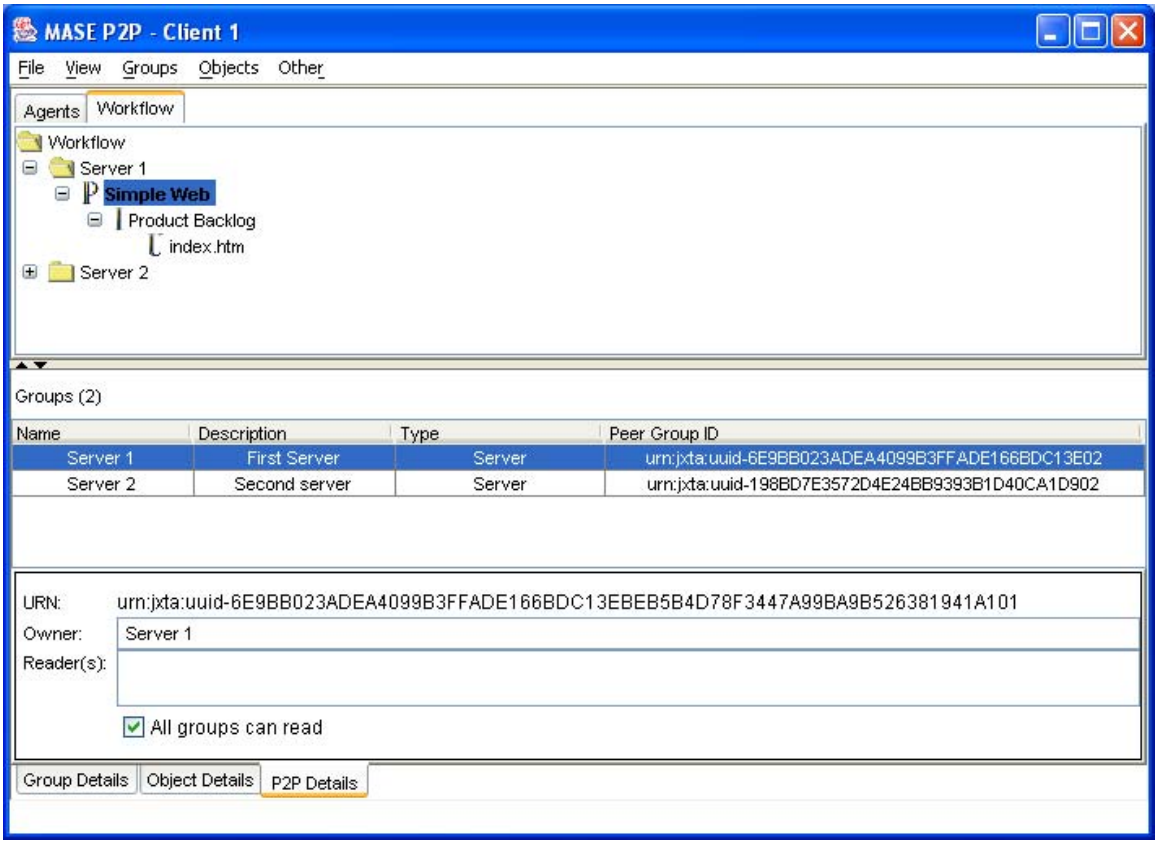

Figure 5-5. The client MASE P2P GUI. The GUI contains three main sections from top to bottom. The top section shows the MASE objects, which in this screenshot is the workflow breakdown. The top section has tabs that allow the user to choose which MASE objects he or she wishes to manipulate. In the middle section, there is a table that displays information about the peer groups that the user is able to communicate with. The bottom section contains tabs for viewing specific details about peer groups or MASE objects.

For more information on the implementation details of the user interface, please refer to Appendix D: GUI Development.

5.5.1 Representation of MASE Objects

MASE objects are represented in such a way that the user can quickly browse through many objects to find objects of interest, while at the same time being able to access detailed information about each object.

Agent objects are represented in a table that can be sorted by column in ascending or descending order. The most important information is only shown in the table in order to keep the size of the table small.

A workflow is a hierarchical structure and so workflow objects were represented in a tree (top section of Figure 5-5). Identifiable icons were used to differentiate between the different types of workflow objects: projects, iterations, user stories, and tasks. Since the objects are related to one another in a hierarchy (i.e., parent-child relationships), P2P information only needed to be associated with some objects and then the children adopt the parent's properties. Those objects that had associated P2P information could be identified because their labels were bolded. Different coloured text is used to differentiate the labels for objects that have different P2P information. For example, if a project is readable by every peer group, but then an iteration within the project is only readable by the owner peer group then the label for this iteration would be blue instead of black. Thus, the user can quickly and easily determine from just looking at the label of the object (and not its details) whether it has different P2P properties than its ancestors.

5.5.2 Menus

The menus in the client GUI contain all of the operations available to the user.

- The *File* menu allows the user to save changes made to the P2P information of MASE objects, reload the preferences from the file system, and exit the program.
- The *View* menu provides options for selectively viewing certain types of peer groups and MASE objects. For example, the user can choose to view only server peer groups, or MASE objects with P2P information.
- The *Groups* menu contains options for retrieving peer group advertisements for peer groups. Once a peer group advertisement has been retrieved, it is shown in the Groups table (i.e., the second table in Figure 5-5). A peer group advertisement must be available in order to communicate with that group.

- The *Objects* menu contains operations for retrieving MASE objects, searching, adding or removing P2P information from objects, and copying objects from one server peer group to another.
- The *Other* menu contains options for viewing help information about the program, and seeing the software's version number (i.e., *About*).

5.6 Functionality

It is difficult to describe all of the functionality present in MASE P2P without viewing a demo of the actual program; nevertheless, some of the functionality is discussed here in order to give an idea of how the users are able to share knowledge, track development, and maintain control of their IP.

5.6.1 Advertisements

Before a user can access information from a group, it must first be able to identify the group, and be able to communicate with the group. Information that allows a group to be identified, such as id, name, and description, is contained within a file called an advertisement. There are different types of advertisements. One type of advertisement has information that is needed in order for a user to connect to and retrieve information from a group.

For a user to communicate with a peer group, it must first obtain the peer group advertisement (PGA) and module specification advertisement (MSA) for the peer group. The PGA contains the peer group id (i.e., the URN), name, description, and MSA id. The MSA includes the MSA id and the pipe advertisement, which defines the pipe type and id, which is used for communicating with the peer group. The user can obtain PGAs from several sources.

- *Local store*. When the user runs the client program, a peer group is automatically created. This peer group listens for other peer groups that join the JXTA network. Thus, the local store contains the advertisements for those peer groups that have

joined the JXTA network after this user logged in, and perhaps advertisements that were obtained from remote peer groups.

- *Remote peer groups.* Advertisements can be obtained from other peer groups. Thus, a user can obtain advertisements for peer groups that joined the JXTA network before it was logged in. For the user to obtain advertisements in this way, he or she must obtain the MSAs for other peer groups through some other means (e.g., e-mail). Once these MSAs are placed into a folder within the MASE P2P client install home directory, the references to these peer groups are loaded automatically when the client program is started. Or, the user can choose to refresh the preferences if the program is already running by selecting the *Reload Preferences* menu item under the *File* menu. It would be beneficial for the MASE peer groups if at least one peer group was always known to be online so that other peer groups could always obtain advertisements for peer groups that are already online.

The user can also choose to access advertisements from both the local store and all of the remote peer groups. Since each peer group has a unique identifier (i.e., a URN), duplicate advertisements can be identified and filtered. The MSAs have a version tag that allows groups to recognize more recent versions of these types of advertisements.

5.6.2 P2P Information for MASE Objects

Information in MASE can be viewed as individual objects, as described in section 4.3.1 Units of Information. A MASE object has two types of information associated with it: MASE information and P2P information. The MASE information is the information that is part of the existing MASE system, which can be viewed using a standard Web browser. The P2P information is data that is used for the MASE P2P functionality. This data is accessed via the MASE P2P client GUI. Note that much of the MASE information is also accessed via the MASE P2P client GUI, which allows a user to identify the MASE objects, and to be able to understand the relationships between workflow objects.

A MASE object only has MASE information associated with it until it is “wrapped” with P2P information (see Figure 5-6), at which point the object can be shared between different groups.

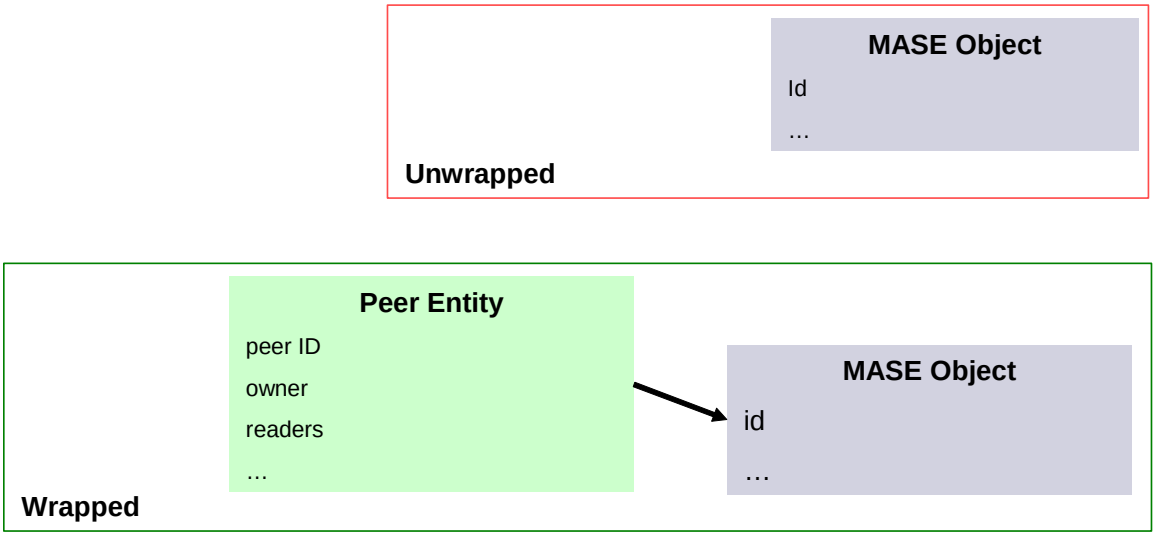

Figure 5-6. An unwrapped MASE object has only MASE information, whereas a wrapped MASE object also has P2P information.

There are the following types of P2P information.

- *Peer ID.* The peer ID is a unique identifier (i.e., URN). It is unique in both time and space. Most IDs are unique in time because otherwise they may no longer be unique if enough time passes. It is especially important for entities (e.g., peers, peer groups, or any object) in a P2P environment to be unique in space because they are not tied to a specific location, and so the identifier must be unique in all locations. The peer ID is a unique 64-bit sequence.
- *MASE ID.* The MASE ID is a unique identifier generated by a MASE server to uniquely identify different types of MASE objects, such as agent, projects, or processes. Thus, this ID is only unique for a particular MASE object type at one server. If a MASE object is copied from one server to another then the MASE ID must be regenerated for the new location.

- *MASE type*. The MASE type represents the type of MASE object, which is one of the following: agent, project, or process.
- *Owner*. The owner is the peer group that has ownership of the MASE object in question. Only a user with an account in the owner peer group can change the ownership of a MASE object.
- *Readers*. A set of peer groups whose agents have read access to the MASE object.
- *Everyone Reader*. If this flag is true then all peer groups have read access to the MASE object.
- *Timestamp*. A timestamp is used to indicate when the P2P information was last updated. This timestamp is only determined based on the clock of the local machine. The timestamp would be most beneficial if it was retrieved from a trusted third-party so that all timestamps were derived from a single clock; however, the drawback is that there would be only one location to retrieve the timestamp. Alternatively, MASE P2P servers could just periodically synchronize their clocks with a central server.
- *Peer Type*. The peer type identifies whether the MASE object is the original or a copy.

The window that allows a user to change the P2P information for a MASE object is shown in the bottom section of Figure 5-5.

5.6.3 Accessing MASE Objects and Updating P2P Information

When using the MASE P2P client GUI, users must have the correct authorization to view MASE objects and update their P2P data. For example, a user can only update the P2P information for a workflow object if the user is an agent in the peer group that has ownership of the object, and the user is a member of the project team. The authorizations required by the user for doing certain operations on both wrapped and unwrapped objects are outlined in Table 5-1.

Table 5-1

Table 5-1. The authorization required by users for doing operations on objects in MASE P2P.

Operation	Object Type	Has Authorization
Retrieving	Unwrapped	- A user with an account in the server peer group. In other words, the user is an agent in the server peer group. For workflow objects, the user must be a member of the project team.
Retrieving	Wrapped	- A user who is an agent in the owner peer group. - A user who is an agent in one of the reader peer groups.
Copying	Wrapped	- A user who is an agent in one of the reader peer groups.
Changing peer information (wrapping, unwrapping, updating P2P information)	Wrapped	- A user who is an agent in the owner peer group. A user can only do one of these operations on his or her agent profile. For workflow objects, the user must be a member of the project team. - A user who knows the wrap password for the peer group.

The user specifies within an XML preference file on his or her local machine the server peer groups for which the user has an account (i.e., is an agent). The user includes a login ID and password for each of these accounts. The user can also specify for each peer group a P2P administrator password, which gives the user the ability to update the P2P information for any MASE objects in a server peer group regardless of whether the user is an agent in the group or a member of the appropriate project team. A preference file was used to hold this information so that if a user performs an operation with several peer groups, the user is not prompted to enter login information for each peer group. As well, the user can include the JXTA login information in the preference file so that there is no prompt to enter a login ID and password whenever the client or server programs are started.

5.6.4 Copying MASE Objects

MASE objects can be copied from one server peer group to another. This functionality allows data to be duplicated and brought closer to users who wish to access the information. For example, let us assume that a user has an account with peer group A (or in other words, is an agent in peer group A), and it wants to copy an object from peer group B, with which it does not have an account. Peer group B is therefore the owner peer group for the object. Let us assume that the object has already being wrapped, and that every peer group can read the object. The first step is for the user to retrieve (or access) the MASE object(s) from the owner peer group. The user can do this because he

or she is an agent in a reader peer group for the object. The second step is for the user to choose to copy the object(s) to group A.

When objects are copied, the user is asked to attach a label to either the first name of an agent object, or the name of a workflow object. For workflow objects, there is a pre-defined structure that must be adhered to in MASE. Thus, if the user chooses to copy one or more workflow objects below the top of the hierarchy, which is a project, then empty placeholder objects must be created in order for the information to be accessible via the MASE user interface. The names for the placeholder objects will by default have the names of the objects they are replacing in the system, but the user will have the option of changing these names in order to conceal information. A project also has a manager, and so the user can either choose an existing agent to be the manager (which may be him or herself), or specify a new user name (i.e., login ID) for a new agent to be the manager. Any workflow objects that are copied will initially be part of projects where only the manager has visibility of the objects. The manager can then choose to let other team members access the information.

5.6.5 Changing Ownership of MASE Objects

A user can change the ownership of MASE objects if it is an agent in the owner peer group and in the peer group that will become the owner peer group. If an object has its ownership changed from group A to group B, then the data is copied to group B where it is identified as being the original, and the instance in group B is now identified as being a copy. This functionality allows a group to assign responsibility for tasks to a different group.

5.6.6 Searching

A search can be done for agent objects. The user has the option of using a regular expression to search for information in any of the fields associated with an agent's profile. This functionality allows a group to discover people outside of their group who may be able to fill the role in one of their projects.

6. LESSONS LEARNED

This section contains some lessons learned during the development of the MASE P2P tool.

6.1 Architecture

Deciding the most appropriate architecture for a program can be one of the most difficult decisions to make. It is often the case that is can be difficult to alter a program from one architecture to another since the implementation and organization of many entities in the program may be affected.

Initially, it was thought that users would interact with the system using HTML, which can be viewed using a standard Web browser (e.g., Internet Explorer). The MASE interface makes use of servlets and Java Server Pages (JSPs) to generate HTML, and so this interface is accessible using a standard Web browser. Using an HTML interface is convenient for users because most computers come with Web browsers pre-installed. However, there were two limitations with using HTML as an interface that I found significant enough to use Java's Swing API instead.

1. The difficulty in representing complex data structures.
2. Reloading, or refreshing information, had to be initiated by the client.

Firstly, using HTML to represent complex data structures such as hierarchical or tree-like structures is cumbersome. The workflow in MASE is represented by a hierarchical structure. Trees are often difficult to represent using HTML because they may can contain too much information than will fit on one page, and it may be difficult for the user to interpret the information in a large tree. Javascript can be used to show and hide tags; however, this developer felt the complexity of representing a tree-like structure in

HTML was much more difficult than representing the same structure using a Java Swing object.

Secondly, when information is sent using HTTP from the server to be displayed in a client's Web browser, the entire page must be reloaded. There may be a delay of seconds if a complex page has to be reloaded to show new information. An HTML page must have finished loading before a user can perform an action, which means a client may have to wait a long time to do an action. Also, refreshing information on the client-side can be problematic if there is information that was generated asynchronously on the server. For a user to obtain new information from the server, a user must either manually choose to reload a page by clicking a button or link, or, there can be a defined period of time before a page is automatically reloaded. A certain amount of responsibility is placed on the user to have to initiate the reloading of a page. If the user forgets or chooses not to reload a page then he or she may miss out on getting access to relevant or important information. Alternatively, if a page is reloaded automatically, a reload may be done unnecessarily, which could lead to an unnecessary interruption and delay for the user. Or, the user may miss out on pertinent information because the reload is done at an inopportune time. Additionally, there is a waste of bandwidth because redundant information is sent over the network.

For these reasons, the GUI was implemented using the Java Swing API because it included classes for all of the types of widgets needed in the user interface. An applet can be implemented and used with the Swing classes.

6.1.1 Applet

A basic applet was implemented to test out the technology, and to check whether there would be any problems in using an applet for the functionality. There were a few problems encountered when using an applet.

1. There was the same reloading problem described above because the applet interacted with the server using HTTP. There would likely have been problems with firewalls if ports other than port 80 had been used.
2. Servers may have to act as middle-men when clients request information.
3. The applet would not be able to get access to files, or write files, on the client machine unless a policy file was changed, or signed applets were used.

Firstly, an HTTP connection was used for the connection between the applet and the server because HTTP connection management is handled automatically by a Web server (e.g., Jetty or Tomcat). Furthermore, it was thought that there may be difficulties using RMI if the user was behind a firewall, because of the inability to access certain ports, or the need to set proxy server properties [RMI2003]. EJBs can be accessed from an applet, but the connections also depend on RMI. Similar to the disadvantages of using the architecture where the user interacts with the system using HTML, using HTTP transactions with an applet means that transactions could only be initiated from the client.

Secondly, one machine (i.e., server) would need to act as a middleman for all of the transactions initiated by the clients that are logged in to the group (see Figure 6-1). There is a burden placed on a machine having to deal with requests from several users and waiting for responses from several other machines. Congestion could develop at these servers for two reasons: 1) the number of clients making requests, 2) the length of time required to fulfill the requests. There could be many users connecting to a server in a peer group. There are rarely guarantees of a machine's up-time in a P2P network, and so a request may need to be timed-out after a defined period of time. During this time, Server A may have to deal with periodic requests from the client asking whether its information has arrived yet. Or, Server A would have to maintain the connection to the client until the response from Server B arrived. Either way, a mechanism would need to be implemented to match the responses from other machines to the user's requests. This approach goes against the P2P idea of having machines interact directly with one another.

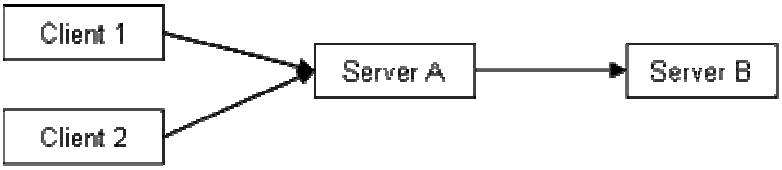

Figure 6-1. An example of the applet architecture. If clients 1 and 2 want to see information on server B, but they only have an account with server A, then they must use server A as a go-between. This architecture can cause a bottleneck at server A.

Thirdly, by default an applet cannot read or write files on a user's machine (i.e., the machine on which the applet is downloaded). This restriction can be circumvented if the user is able to alter the JVM policy file, or is comfortable dealing with a signed applet.

6.1.2 Application Architecture

The third architecture considered was an application architecture, where each user runs a standalone application to connect to the MASE P2P network. A standalone application is run on both the client and server machines. Because of the way in which the JXTA technology works, each instance of the application has its own JXTA peer group. However, an actual team of developers is represented by the users who have accounts on a MASE server. Thus, an abstract team could be represented by users who have accounts on more than one MASE server (Figure 6-2).

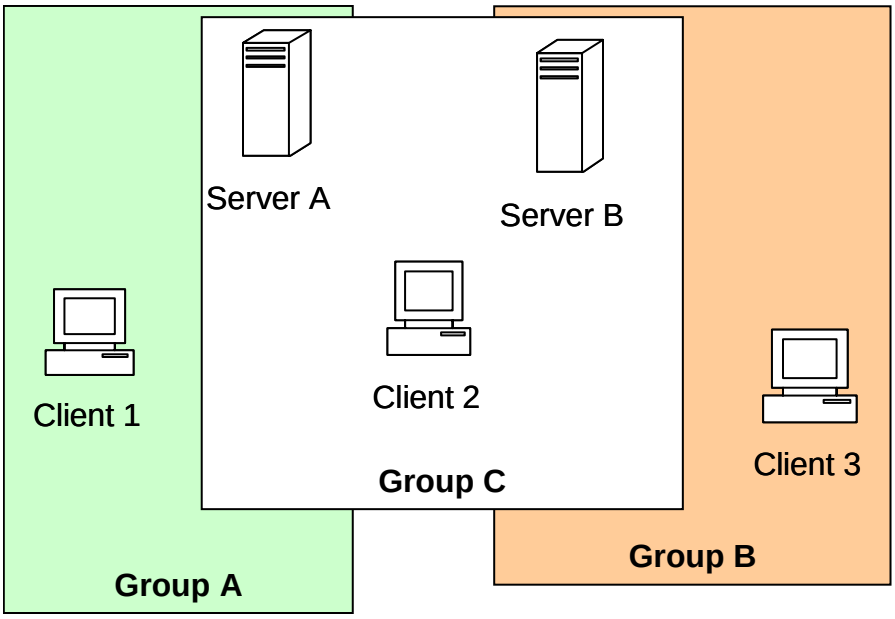

Figure 6-2. Client 2 has an account on servers A and B, which means client 2 can be thought of as being part of an abstract group C.

This is more akin to a P2P architecture because each both the client and server applications act as peers in the JXTA network, rather than just the servers acting as peers. Thus, both the client and server applications can communicate directly with one another, which means the client and server machines have more independence and share the workload in the network.

The application architecture has the following benefits.

1. *Servers only handle requests for local information*. A server only needs to retrieve information from its local database to respond to a client request. The result is that the servers handle fewer requests than the applet architecture (see Figure 6-3).
2. *Servers can initiate transactions*. The server can initiate a transaction and send data to the client once the client becomes available. The connection between the client and the server does not need to remain open, and the client does not need to query a server more than once for each request. On the client side, once the data arrives it can be made available to the user immediately and to the appropriate

location in the GUI depending on the type of data. All of this can be done without the user being interrupted since threads can be used to listen for incoming data.

3. *Files can be read and written on the client machine*. Files can be read and written on the client as well as the server, which means that each client can have its own local preferences, and the client is able to store a repository of data. For example, a client could store the workflow for the task within a project.

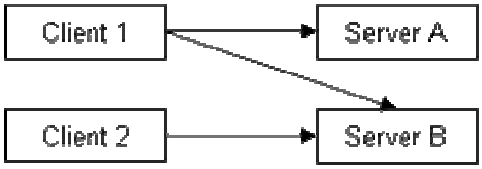

Figure 6-3. An example of the application architecture. Clients can connect directly to any servers.

The main disadvantage to this approach is that users that are acting as clients must download and install the software on their own machines. However, because the program runs using the JVM, the installation could be as simple as only having to decide where to extract the contents of an archived file.

Another disadvantage is that there is a separation of the MASE UI from the MASE P2P UI. It is arguable that the user would be able to more easily understand and pick up the functionality if the same UI was used. Regardless of the aforementioned disadvantages, the application architecture was used because in the author's opinion the benefits of using this architecture that supported the P2P architecture the best outweighed the disadvantages.

7. EXAMPLE OF USING MASE P2P

This section explains how the MASE P2P tool could be used in a real-life scenario. The example of using MASE P2P is a continuation of the math functions Web service project described in section 3.2 Example Scenario. Company A is heading a project to implement a Web service that computes math functions. Company B is hired to implement the business logic, but it wants to have a tight control over the development process and information used because it wants to use its own development process and it wants to make use of code from other projects. In the end, company B is only delivering binary code. The Web service code is dependent upon the business logic, and so project tracking is used to provide an efficient means for company A to keep up to date on the progress of company B so that the services can be made available as soon as possible.

First of all, company A and B need to both have computers that support the Java VM (i.e., Windows, Mac OS, or Linux). Each company must install and deploy JBoss, MySQL, MASE, and the MASE P2P server software on the same machine. The two companies must alter their peer group advertisement files in order to identify the names and descriptions of their peer groups. Starting the MASE P2P server software will log the two peer groups onto the network. The MASE P2P client software can be installed on different machines within the companies.

Once the necessary software is installed, a user (e.g., John) in company A can create a new project and workflow breakdown by logging in to MASE using a Web browser. The workflow would contain the tasks for implementing the business logic and the Web service interface code. Figure 7-1 contains a high-level workflow breakdown of the project and the tasks that each company is responsible for implementing. John could then assign ownership of the Math Functions task to company B, and set the manager of this task to be Mary, who is a developer in company B.

Company B now has control of the Math Functions task and all tasks below it. Mary can then add sub-tasks to the tasks of implementation functions 1, 2, etc. In order for the users in company A to be able to track the progress of the functions being implemented, Mary must allow company A to have read access to some of the tasks. In order for company B to be able to protect the details of its implementation while at the same time allowing company A to track the progress of the development, company B would only give read access to the highest-level tasks. A member of company A's peer group can now access the details of these high-level tasks using the MASE P2P client user interface in order to determine how much remains in the implementation of a function.

The MASE P2P tool offers allows company A to better manage the Math Web Service project by using project tracking to determine the status of development of code that is being developed by a remote team from another organization (i.e., company B).

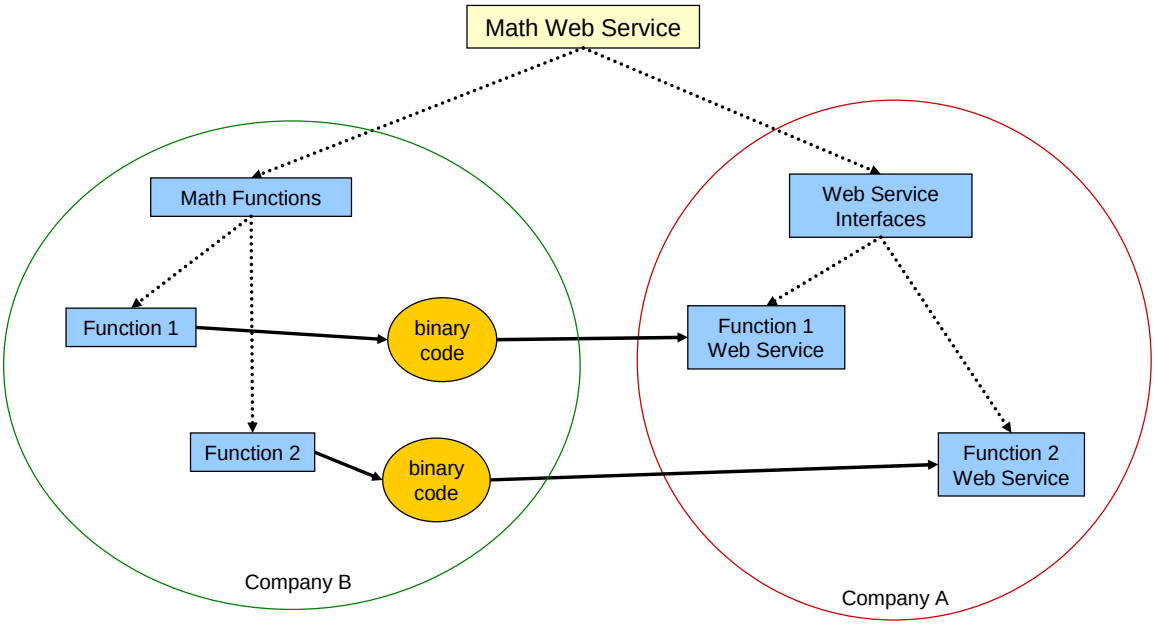

Figure 7-1. The high-level workflow breakdown for the Math Web Service project. Company A is responsible for implementing the math functionality (i.e., the business logic) and company B is responsible for implementing the Web service interface code.

8. QUALITATIVE ANALYSIS

The purpose of this section is to provide a qualitative analysis of the value of the software development support derived from using a P2P architecture, and the MASE P2P tool itself. The value will be based on whether the requirements were met, and how the tool compares to other comparable tools.

The requirements were chosen based on the research goals. The research goals for this work were the following:

1. To develop a flexible solution that allows teams within virtual corporations to balance the control and sharing of their IP in distributed software development projects.
2. To examine the design issues for a tool, which is based on a P2P architecture.
3. To implement a tool as a proof-of-concept.
4. To do a qualitative analysis of the approach to assess its merit and address any issues.

The first goal was to address the limitation that current software development tools have when it comes to protecting IP by developing a solution that allows teams to flexibly manage their IP. Current process-support tools use a client-server architecture, which means that teams working together in a virtual corporation must relinquish control of their data in order to make use of collaboration support. Using a P2P architecture allows a group to store its data locally, which means it does not have to relinquish control to another group running a central server. While at the same time, a group can collaborate with other groups by deciding what information it wants to share with others.

The design issues were discussed in detail in section 4 MASE P2P Design. Some of the design decisions dealt with assigning roles to different groups in order that access privileges to data could be enforced. Only users that are part of a designated group would

be able to access certain information. Groups that could access data could also make copies of information. A hybrid push and pull solution was proposed if data synchronization was going to be implemented. A group could assign responsibility for tasks to another group, and then the development of these tasks could be tracked. Thus, the tool supports the flexible management of IP because knowledge can be tightly controlled or shared during collaboration.

The functionality for the tool was implemented with the requirements discussed in chapter 3 MASE P2P Requirements. The MASE P2P tool requires only a few steps for the installation. The user interface is simple and intuitive, and the user can do a single operation on several objects at one time, which saves the user time from doing repetitive tasks.

The tool allows groups from different locations to share member profile and workflow information according to the rules of access they specify. Groups running MASE servers can connect to the network from any location where there is an Internet connection in order to make available agent profiles and workflows for other teams to acquire. Agent profiles are searchable, which allows users to find agents based on any of the information in their profile, such as an agent's skills. Groups are able to connect to the network and immediately make available any information they wish to share with others.

The tasks in a project can be divided between different teams and each team can control the information that other groups are able to see. A team is able to make use of an assigned task structure and alter the workflow according to its own needs. A team can use MASE to plan and track tasks during development, and can make use of additional group information for development without having to relinquish control of this intellectual capital to users outside of their group.

Groups can store their data locally because the tool is run in a P2P network with the use of the JXTA technology. The tool benefits from the characteristic advantages of being a P2P application, including the opportunity to easily introduce redundancy into the network, and have network load balanced across several machines.

8.1 Comparison with other Tools

MASE P2P is a tool that supports distributed software development using a P2P architecture. Because the purpose of this research was to address some limitations with the current software development tools, only those tools that provided similar software development support (e.g., process support) were considered. For example, the many P2P applications that only support file sharing were not included in this comparison. And, only software development tools that were mature enough to be actually used were considered. It is important to note that MASE by itself offers a fair amount of support over other tools [Bowen2002a]. Because MASE P2P is only an extension of the MASE tool and cannot be used independently, the features of two are considered together as one application.

- *CodeWright*. This is a product released by Borland that allows developers to communicate and do file editing and sharing using P2P connectivity [CodeWright2003]. This tool would be useful for developers doing pair programming, but it does not provide much support for project management.
- *Groove*. Groove Networks offers several P2P applications that are geared towards allowing people to collaborate and synchronize information in shared workspaces. One of these applications is the TeamDirection Project [TeamDirection2003], which is part of the Groove Workspace Project Edition [GWPE2003]. It is a tool with a refined user interface that allows team members to create and manage tasks in a Gantt work breakdown. Furthermore, files and projects can be synchronized across workspaces. Groove does not support the same level of flexibility as MASE P2P when it comes to setting privileges for accessing information.

Permissions can be set for different members based on their roles (i.e., manager, participant, and guest). However, they are set for an entire project and cannot be applied to specific tasks within a project. Because permissions cannot be set at the task-level within a project, the workflow for a project cannot easily be divided and implemented by different groups who want to set their own permissions.

- *Rational Unified Process*. The Rational Unified Process (RUP) platform provides guidance for doing development using a variety of approaches. Some of the processes supported include XP, business modeling, and systems engineering [RUP2003]. The platform provides a great deal of information and resources for the development, and tools for generating report-like documents, but it does not provide support for creating an actual dynamic workflow breakdown with tasks that could have an active role (e.g., informing an agent that he or she has been assigned a task).
- *VistaPoint.* The VistaPoint tool is built upon the JXTA platform [VistaPoint2003]. It provides a P2P framework for doing data exchange and synchronization. However, it does not support the planning and enactment of project development.
- *Microsoft Project*. This tool provides a comprehensive set of features that covers collaboration, building project schedules, and synchronizing information across team members [Project2003]. It seems to include most features that are needed for managing a project. Although it does support setting up servers for users to retrieve information, it does not support a user being able to move information from one server to another.
- *VersionOne*. The VersionOne tool is a project planning and management tool for agile development [VersionOne2003]. It also supports collaboration, but it is not based on a P2P architecture and so data must be stored centrally.
- *TUKAN*. This tool provides communication and awareness support for teams doing software implementation [Schümmer2000]. As developers do implementation, they are made aware of possible conflicts in code that they are

working on. A developer can easily chat with the other developers to resolve any conflicts. There is also a history of changes that is stored in the system that is shown to a developer when he or she begins work.

- *GENESIS*. This tool aims to be a flexible and distributed environment for modeling and controlling software engineering processes, and to support the management of co-operation among team members [GENESIS2003]. It has support for managing project workflows and the artefacts that are produced from tasks.
- *CAGIS PCE*. The CAGIS process centred environment (PCE) is a tool that combines an activity-based workflow with software agents in order to better support cooperative activities among developers [Wang2002].

A summary of these tools and MASE P2P is included in Table 8-1.

To summarize, the P2P applications (i.e., CodeWright, Groove, VistaPortal) were generally weaker in terms of the process support for distributed software development. Groove did have process support, but it only allowed access privileges to be defined at the project level, and not at lower levels such as the task level. The tools also did not include support for finding members based on profiles. The tools geared towards distributed software development and process support had the limitation of not supporting the local storage of information because they were built on a client-server architecture. Thus, the management of IP in these tools is not as flexible as a tool built on a P2P architecture, where teams can store their development data locally and maintain tight control of the information. This was a limitation that served as a motivation for the research and development of MASE P2P (see section 3.1).

MASE P2P includes features for process support, knowledge sharing, and the flexible management of IP. One feature that was common in several of the other applications and would be useful in MASE P2P is the ability to synchronize data across multiple sites. The

design issues related to this functionality are discussed in detail in section 4.5.2 Data Synchronization.

Table 8-1. A comparison of software development tools.

Tool	**Distributed**	**Collaboration**	**Sharing Member Profiles**	**Project Tracking**	**Access Control**	**Local Storage (i.e., P2P)**	**Copy Data**	**Strengths and Weaknesses**
MASE P2P	yes	yes	yes	yes	task level	yes	yes	strengths: distributed software development, P2P weaknesses: no data synchronization
CodeWright	yes	yes	no	no	no	yes	no	strengths: remote coding and editing weaknesses: process support and project management
Groove	yes	yes	no	yes	project level	yes	yes	strengths: numerous project management features weaknesses: access control only at project level, member profile is only member's name
Rational Unified Process	yes	no	no	no	no	no	no	strengths: guidelines for different development processes weaknesses: no actual tool support during development
VistaPoint	yes	yes	no	no	no	yes	yes	strengths: several tools for visualizing and working with artefacts weaknesses: no support for workflow management
Microsoft Project	yes	yes	yes	yes	project level	no	yes	strengths: large set of development support weaknesses: reliance on central servers
Version One	yes	yes	no	yes	project level	no	no	strengths: distributed software development weakness: dependency on central server
TUKAN	yes	yes	no	no	no	no	no	strengths: conflict awareness
GENESIS	yes	yes	unknown	yes	project level	no	no	strengths: flexible process support weaknesses: dependency on a central server; project-level access control
CAGIS PCE	yes	yes	unknown	yes	no	no	no	strengths: flexible cooperative and process support weaknesses: dependency on central server

9. CONCLUSION

Software projects may be developed by several teams for financial or practical reasons. Software tools allow teams to collaborate and coordinate their tasks even when they work in different locations, such as different building, cities, or even countries. A distributed environment allows developers to form virtual teams and corporations. A virtual team may include developers from many different organizations who have specific skills that are required in a project.

Even though teams may be working on relatively self-contained and cohesive modules, some transfer of knowledge is required between the teams. There are almost always some dependencies, and at some point the modules must be incorporated together. As well, communication is needed for the overall coordination and management of the project; for example, the tracking of project milestones. It is beneficial for a team to filter the information that it makes available to other teams because not all of the information is likely important or necessary. Furthermore, a team may want to make use of confidential information within their own group when developing a module for a multi-team project (i.e., in a virtual corporation). This information may be helpful in the development of the module, but it will not be part of the deliverable. For example, if only binary code or a service is being delivered then a team has little to worry about inadvertently releasing company secrets. In other words, a group's IP could be compromised if the information being made available outside the team is not filtered.

Teams can flexibly manage their IP if they use a tool that is built on a P2P architecture. Teams can collaborate with one another and still have the freedom to maximize control of their IP because they are able to store data locally. Most of the tools that support distributed software development have been based on a client-server architecture, where some machines spend most of the time retrieving information from a machine that stores all of the data for a project. A P2P architecture allows for flexible relationships between

machines, where machines can adopt the roles of both server or client. The P2P model also provides additional advantages for a system, including load balancing and more robustness in the network

A P2P tool called MASE P2P was developed that supports the inter-team development of software in a distributed setting, while at the same time allowing teams to have a high-level of control over their IP. MASE P2P is built upon the MASE tool, which includes the following support for distributed projects: knowledge management of developers and their skills, coordination of tasks during development, and XP project planning and management.

The design of the system included considerations of the detailed architecture, what information will be shared, how the information will be shared, and the security measures required in order to enforce the authentication of users and the access privileges of data. The tool makes use of the JXTA technology, which is a set of protocols that allows computers running the JXTA implementation to connect with any other computer regardless of platform or network.

The functioning tool is a proof of concept that a tool can be used to address the limitations with using a client-server configuration for a tool that supports distributed software development. The tool is open-source and the code freely available (see Appendix A: MASE P2P Site), which means that the tool can be built upon by others, which can lead to further advances in this research area.

The MASE P2P tool uses a P2P architecture to provide software development support that would not otherwise be possible with a traditional client-server architecture. The most novel benefits over comparable software development tools is that the tool offers the benefits of being based on a P2P architecture, and it provides a simple method for teams to share (i.e., replicate) knowledge across team boundaries. Most importantly, a

project can be divided among several sovereign teams that are able to use whatever means they wish to do the actual development, which may include the use of existing proprietary or sensitive company information in the process. At the same time, teams have support for collaboration because information can be made available to other developers working on the same project. This is possible because each team can store their information on their own machines within their own network, while part of the information is still integrated into the overall workflow.

The use of a P2P tool in software development is a new idea that certainly seems to provide benefits to the increasingly common distributed nature of software development. This area of research is in its infancy, and so it will be exciting to see the advances that the P2P model can continue to provide to software engineering and other areas of development.

10. FUTURE WORK

Future work for this research includes future research work and implementation improvements.

It would be beneficial to do an assessment of the merit of the P2P functionality compared to the client-server functionality. Specifically, from a process perspective, does the P2P functionality increase the efficiency of the development process? Performance measures could include measuring the delays when doing certain operations in the program. As well, from a practical perspective, do developers working in inter-team distributed projects feel that they have more control over their IP, and if so, whether it is necessary? Questionnaires could be used to collect data on the developers' impressions.

The current implementation of the tool (version 0.3) allows the user to assign access rights for MASE objects, and then objects can be copied from one machine to another if the user has the appropriate access rights. Thus, the system allows a group to restrict access to information according to its own interest while at the same time being able to make available whatever information it chooses, which may be useful when coordinating work in a project, or for knowledge sharing. There are several implementation improvements that could be made to the system that would provide further support for teams wanting to collaborate on projects.

1. *Automatic data synchronization*. When a copy is made of a MASE object, there is no automatic mechanism that will update copies if an original changes. In the current version, a user could manually check whether an original instance has changed because copies specify the owner (i.e., which group has control of the original) and unique id of the original instance. An automatic update would save the time required to do the retrieval and comparison. An event notification system could also be used to inform groups that their copies require updating.

2. *Service Availability*. It would be helpful for users if peer groups had some service availability or quality of service information associated with them. For example, a user could find out how often the group is online, or the speed at which data can normally be downloaded.
3. *Searching for workflow objects*. It is possible to search for agents in the network using any of the fields in an agent's profile. However, there was not enough time to implement searching for workflow objects (i.e., projects, iterations, user stories, or tasks). Teams could certainly benefit from being able to search and download a workflow that could serve as a template for the development of a similar project.
4. *Improved user feedback*. A user receives little feedback outside of the new data being incorporated into the appropriate window when requests to several servers are successful. For example, if a request is made to copy a workflow object from one server to another then the only way the user is able to see that the operation succeeded is because the workflow tree structure collapses and no error messages appear. However, if the tree is already collapsed and no reply arrives then a user will not know whether the operation was successful or not. An optional view could show the outstanding requested operations and the returned messages in a table. This approach would prevent the user having to look at several modal windows that pop up. Additionally, the user could choose to show only error messages (i.e., have messages removed automatically for successful operations).
5. *Clients storing MASE data*. If a client could store information locally then users could form peer groups composed entirely of lightweight devices. As well, MASE objects could be copied to machines only running the client software, which would allow users to make use of some of the MASE support without having to maintain a server (i.e., running a DBMS and J2EE application server).
6. *Messaging between peer groups*. A messaging system would allow peers to send each other messages as a form of asynchronous and synchronous communication. Even if a peer group was not online, then the message could be stored at a server that is always online. When a peer group goes online, a request could be done to

this server to retrieve any messages. The server could serve the same role as a mail server, but could also be used for storing information about updates.

7. *Negotiations for ownership*. Negotiations are not supported when a group wishes to assign ownership of an object to another group. It is assumed that negotiations take place through some other means, such as e-mail or instant messaging. Including negotiations in MASE P2P would be a convenience feature for the user. Messaging could be used for negotiations.

11. REFERENCES

[Aberer2002] Aberer, K., Punceva, M., Hauswirth, M., and Schmidt, R., "Improving Data Access in P2P Systems," *IEEE Internet Computing*, January-February 2002, pp. 58-67.

[AgileManager2003] Agile Manager, http://xpmanager.sourceforge.net/, accessed September 11, 2003.

[Akamai2003] Akamai: How it works, http://www.akamai.com/en/html/services/edgecomputing_howitworks.html, accessed September 22, 2003.

[Anonymizer2003] Anonymizer – Online Privacy and Security, http://www.anonymizer.com/, accessed September 25, 2003.

[Apache2003] Apache HTTP Server Project, http://httpd.apache.org/, accessed July 24, 2003.

[Balakrishnan2003] Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., and Stoica, I., "Looking up Data in P2P Systems", *Communications of the ACM*, vol. 46, no. 2, February 2003, pp. 43-48.

[Barkai2001] Barkai, D., "Technologies for Sharing and Collaborating on the Net", *Proceedings of the First International Conference on Peer-to-Peer Computing*, 2001, pp. 13-28.

[Beck2000] Beck, K., *Extreme Programming Explained: Embrace Change*, Addison Wesley, Upper Saddle River NJ, 2000.

[Botros2001] Botros, S. and Waterhouse, S., "Search in JXTA and Other Distributed Networks," *Proceedings of the First International Conference of Peer-to-Peer Computing*, 2001, pp. 30-35.

[Bowen2002a] Bowen, S. and Maurer, F., "Process Support and Knowledge Management for Virtual Teams Doing Agile Software Development," *Proceedings of the 26th IEEE Annual International Computer Software and Application Conference*, 2002, pp. 1087-1092.

[Bowen2002b] Bowen, S. and Maurer, F., "Designing a Distributed Software Development Support System Using a Peer-to-Peer Architecture," *Proceedings of the 26th IEEE Annual International Computer Software and Application Conference*, 2002, pp. 1118-1120.

[Byrne1993] Byrne, J. A., Brandt, R., and Port, O., "The Virtual Corporation," *Business Week*, February 8, 1993, pp. 36-40.

[Carmel2001] Carmel, E. and Agarwal, R., "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Software*, March-April, 2001, pp. 22-29.

[Cloak2003] The cloak – free anonymous Web surfing, http://www.the-cloak.com/anonymous-surfing-home.html, accessed September 25, 2003.

[Cockburn2002] Cockburn, A., *Agile Software Development*, Pearson Education, Inc, Boston, MA, 2002.

[CodeWright2003] CodeWright Features, http://www.premia.com/products/, accessed November 9, 2003.

[Costello2003] Costello, S., "Webnoize reports Napster downloads drop 36 percent in April," http://archive.infoworld.com/articles/hn/xml/01/05/01/010501hnnapster.xml?p=br&s=8, accessed on October 16, 2003.

[Creedon2003] Creedon, E., Humpreys, D., Kelly, B., Kinane, S., and Elstner, K., "GNUtella," http://ntrg.cs.tcd.ie/undergrad/4ba2.02-03/p5.html, accessed on September 11, 2003.

[Crowston2002] Crowston, K. and Scozzi, B., "Open Source Software Projects as Virtual Organisations: Competency Rallying for Software Development," *IEEE Proceedings of Software*, vol. 149, issue 1, feb. 2002, pp. 3-17.

[CVS2003] Concurrent Versions System - The open standard for version control, http://www.cvshome.org, accessed September 11, 2003.

[Damian2002] Damian, D. E. and Zowghi, D., "The Impact of Stakeholders' Geographical Distribution on Managing Requirements in a Multi-site Organization," *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 319-328.

[Delio2001] Delio, M., “Microsoft’s Web Still Tangled,” *Wired News*, http://www.wired.com/news/business/0,1367,41423,00.html, accessed September 22, 2003.

[Dossick1999] Dossick, S. E., Port, D., and Kaiser, G. E., “Embedding Model-Based Architecting in a Collaborative Environment,” http://www.psl.cs.columbia.edu/ftp/psl/CUCS-016-99.pdf, accessed September 10, 2003.

[Dutoit2001] Dutoit, A.H., Johnstone, J., and Bruegge, B, “Knowledge Scouts: Reducing Communication Barriers in a Distributed Development Project”, *Eighth Asia-Pacific Software Engineering Conference*, 2001, pp. 427-430.

[EBE2003] e-Business Engineering at the University of Calgary, http://sern.ucalgary.ca/~milos/, accessed October 16, 2003.

[Fowler2003] Fowler, M., “Model View Controller,” *Patterns of Enterprise Application Architecture*, Addison Wesley, Upper Saddle River NJ, 2003, pp. 330-332.

[FreeHaven2003] The Free Haven Project, http://www.freehaven.net/, accessed September 12, 2003.

[Freenet2003] The Freenet Project, http://freenetproject.org/, accessed September 12, 2003.

[GENESIS2003] GENESIS User Manual, http://prdownloads.sourceforge.net/genesis-ist/GenesisUserManual_V0_3.zip?download, accessed November 3, 2003.

[Gnutella2003] Gnutella.com, http://www.gnutella.com, accessed September 30, 2003.

[Gong2001] Gong, L. “JXTA: A Network Programming Environment,” *IEEE Internet Computing*, vol. 5, no. 3, May-June 2001, pp. 88-95.

[Gong2002] Gong, L. “Peer-to-Peer Networks in Action,” *IEEE Internet Computing*, vol. 6, no. 1, January-February 2002, pp. 37-39.

[Google2003a] Google, http://www.google.com, accessed September 11, 2003.

[Google2003b] Google Groups Help, http://www.google.ca/googlegroups/help.html, accessed December 9, 2003.

[Groove2003] Groove Networks, Inc., Desktop Collaboration Software, http://www.groove.net, accessed September 12, 2003.

[GWPE2003] Groove Workspace Project Edition, http://www.groove.net/products/workspace/project-edition/, accessed September 30, 2003.

[Harker1992] Harker, S.D.P., Eason, K.D., and Dobson, J.E, "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering," *Proceedings of the IEEE Symposium on Requirements Engineering*, 1992, pp. 266-272.

[J2EE2003] Java 2 Platform, Enterprise Edition, http://java.sun.com/j2ee/, accessed August 28, 2003.

[Jabber2003] Jabber Software Foundation, http://www.jabber.org/, accessed September 30, 2003.

[Java2003] The Source for Java Technology, http://www.java.com/en/index.jsp, accessed September 14, 2003.

[JBoss2003] JBoss: Professional Open Source, http://www.jboss.org/, accessed September 11, 2003.

[Jini2003] Jini Network Technology, http://wwws.sun.com/software/jini/, accessed July 24, 2003.

[JVM2003] Download Java Software, http://www.java.com/en/download/manual.jsp, accessed September 15, 2003.

[JXTA2003] JXTA Project Home Page, http://platform.jxta.org/servlets/ProjectHome, accessed September 15, 2003.

[Kan2001] Kan, G., "Gnutella," in *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, edited by Andy Oram, O'Reilly & Associates, Sebastopol, CA, 2001, pp. 94-122.

[Kazaa2003] Kazaa Lite – Official Website. http://www.kazaalite.tk/, accessed August 7, 2003.

[Kemp2001] Kemp, L.L., Nidiffer, K.E., Rose, L.C., Small, R., and Stankosky, M., "Knowledge Management: Insights from the Trenches," *IEEE Software*, vol. 18, no. 6, Nov-Dec. 2001, pp. 66-68.

[Kim1998] Kim, S.D., Rhew, S.Y., Kim, C.J., and Kim, D.K. "Object Clustering Techniques for Client/Server and Distributed Software Development," *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, Oct. 1998, pp. 2675-2679.

[Kötting1999] Kötting, B. and Maurer, F., "A Concept for Supporting the Formation of Virtual Corporations through Negotiation," *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1999, pp. 40–47.

[Lawrence1998] Lawrence, S. and Giles, C.L., "Searching the World Wide Web," *Science*, vol. 280, no. 5360, 1998, pp. 98-100, available online at http://citeseer.nj.nec.com/lawrence98searching.html.

[Lawrence1999] Lawrence, S. and Giles, C. L., "Accessibility of Information on the Web," *Nature*, vol. 400, 1999, pp. 107-109.

[Leiner2000] Leiner, B.M., Cerf, V.G., Clark, D.D., Kahn, R.E., Kleinrock, L., Lynch, D.C., Postel, J., Roberts, L.G., and Wolff, S., "A Brief History of the Internet," http://www.isoc.org/internet/history/brief.shtml, 2000, accessed September 11, 2003.

[Lethin2003] Lethin, R., "Technical and Social Components of Peer-To-Peer Computing", *Communications of the ACM*, vol. 46, no. 2, February 2003, pp. 30-32.

[Lienhart2002] Lienhart, R., Holliman, M., Chen, Y.-K., Kozintsev, I., and Yeung, M., "Improving Media Services on P2P Networks," *IEEE Internet Computing*, Jan-Feb 2002, vol. 6, no. 1, pp. 73-77.

[Linux2003] Linux Home Page at Linux Online, http://www.linux.org, accessed July 24, 2003.

[Macedonia2000] Macedonia, M., "Distributed File Sharing: Barbarians at the Gates?" *Computer*, vol. 33, no. 8, Aug. 2000, pp. 99-101.

[Maurer2000] Maurer, F., Dellen, B., Bendeck, F., Goldmann, S., Holz, H., Kotting, B., and Schaaf, M., "Merging Project Planning and Web Enabled Dynamic Workflow Technologies," *IEEE Internet Computing*, vol. 4, issue 3, May-June 2000, pp. 65-74.

[McCullagh2000] McCullagh, D., "Hotmail Down Due to Hole," *Wired News*, http://www.wired.com/news/technology/0,1282,36249,00.html, accessed September 22, 2003.

[Messenger2003] MSN Messenger, http://messenger.msn.com/, accessed September 12, 2003.

[Minar2001] Minar, N. and Hedlund, M. "A Network of Peers: Peer-to-Peer Models Through the History of the Internet", in *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, edited by Andy Oram, O'Reilly & Associates, Sebastopol, CA, 2001, pp. 3-20.

[MIT2003] Open Source Initiative OSI - The MIT License, http://www.opensource.org/licenses/mit-license.php, accessed October 5, 2003.

[Mockus2001a] Mockus, A. and Herbsleb, J., "Challenges of Global Software Development", *Proceedings of the IEEE Seventh International Software Metrics Symposium*, 2001, pp. 182-184.

[Mockus2001b] Mockus, A. and Weiss, D.M., "Globalization by Chunking: a Quantitative Approach," *IEEE Software*, March-April, 2001, pp 30-37.

[Mont2001] Mont, M.C. and Tomasi, L., "A Distributed Service, Adaptive to Trust Assessment, Based on Peer-to-Peer E-Records Replication and Storage," *Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001, pp. 89-95.

[MySQL2003] MySQL: The World's Most Popular Open Source Database, http://www.mysql.com/, accessed September 11, 2003.

[Netcraft2003] Netcraft Web Server Survey, http://www.netcraft.com/survey/, accessed July 24, 2003.

[NetMeeting2003] Microsoft NetMeeting, http://www.microsoft.com/windows/netmeeting/, accessed July 24, 2003.

[Nua2003] Nua Internet How Many Online. http://www.nua.ie/surveys/how_many_online/, accessed September 4, 2003.

[Perl2003] Perl Mongers, http://www.perl.org/, accessed July 24, 2003.

[Portmann2001] Portmann, M., Sookavatana, P., Ardon, S., and Seneviratne, A., "The Cost of Peer Discovery and Searching in the Gnutella Peer-to-peer File Sharing Protocol," *Proceedings of the Ninth IEEE International Conference on Networks*, 2001, pp. 263-268.

[Project2003] Project. Microsoft Project Home Page. http://www.microsoft.com/office/project/default.asp, accessed on August 7th, 2003.

[Publius2003] Publius Home Page, http://publius.cdt.org/, accessed September 12, 2003.

[Reid2003] Reid, R, "Canada Trumps U.S. In Broadband Use, comScore Media Metrix Canada Reports," *comScore Software*, http://www.comscore.com/press/release.asp?id=312, accessed September 4, 2003.

[Rein2002] Rein, L., "Peer-to-peer XML," *IEEE Internet Computing*, vol. 6, no. 2, March-April 2002, p. 100.

[Renesse2002] van Renesse, R., Birman, K., Bozdog, A., Dumitriu, D., Singh, M., and Vogels, W., "Heterogeneity-Aware Peer-to-Peer Multicast," Oct. 2002. submitted to 2nd International Workshop on Peer-to-Peer Systems, 2002, http://www.cs.cornell.edu/Info/Projects/Spinglass/public_pdfs/heterogeneity.pdf, accessed September 12, 2003.

[Ripeanu2001] Ripeanu, M., "Peer-to-Peer Architecture Case Study: Gnutella Network," *First International Conference on Peer-to-Peer Computing*, 2001, pp. 99-100.

[RMI2003] RMI and Object Serialization – FAQ, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/faq.html, accessed September 18, 2003.

[Royce1970] Royce, W.W., “Managing the Development of Large Software Systems: Concepts and Techniques,” *Proc. WESCON*, August 1970.

[RUP2003] Rational Unified Process, http://www.rational.com/products/rup/index.jsp, accessed September 30, 2003.

[Salutation2003] Salutation. Salutation Consortium, http://www.salutation.org, accessed July 24, 2003.

[Schümmer2000] Schümmer, T. and Schümmer, J., “Support for Distributed Teams in eXtreme Programming,” http://citeseer.nj.nec.com/372706.html, 2000, accessed September 10, 2003.

[Schwaber2002] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Prentice Hall, Upper Saddle River, NJ, 2002.

[SourceForge2003] SourceForge.net, http://sourceforge.net/, accessed November 6, 2003.

[Stadel2003] Stadel, M. Systems Support, Department of Computer Science, University of Calgary, personal communication in August 2003.

[Swing2003] Creating a GUI with JFC/Swing, http://java.sun.com/docs/books/tutorial/uiswing/, accessed September 28, 2003.

[Tan2000] Tan, B. C. Y., K.-K. Wei, Huang, W. W., Ng, G.-N., “A Dialogue Technique to Enhance Electronic Communication in Virtual Teams,” *IEEE Transactions on Professional Communication*, vol. 43, issue 2, June 2000, pp. 153-156.

[TeamDirection2003] TeamDirection – Project Management for Virtual Teams, http://www.teamdirection.com/tdweb/main/index.php, accessed September 30, 2003.

[Traversat2002] Traversat, B., Abdelaziz, M., Duigou, M., Hugly, J.-C., Pouyoul, E., and Yeager, B., “Project JXTA Virtual Network,” http://www.jxta.org/project/www/docs/JXTAprotocols.pdf, 2002, accessed September 16, 2003.

[VersionOne2003] VersionOne Features and Benefits, http://www.versionone.net/pdf/V1%20Features%20and%20Benefits.pdf, accessed November 3, 2003.

[VistaPoint2003] VistaPoint, http://www.vistaportal.com/products/vistapoint.htm, accessed November 9, 2003.

[Wang2002] Wang, A. I., "A Process Centred Environment for Cooperative Software Engineering," *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, 2002, pp. 469-472.

[Waterhouse2002] Waterhouse, S., Doolin, D.M., Kan, G., and Faybishenko, Y., "Distributed Search in P2P Networks," *IEEE Internet Computing*, vol. 6, no. 1. Jan-Feb. 2002. pp. 68-72.

[Wilcox2003] Wilcox, J., "Microsoft Expands Rights Management Tool", *CNET News.com*, http://news.com.com/2100-1001_3-985496.html, accessed on September 26th, 2003.

[WSAD2003] WebSphere Studio Application Developer - Product Overview - IBM Software, http://www-3.ibm.com/software/awdtools/studioappdev/, accessed September 25, 2003.

[XML2003] XML in 10 points, http://www.w3.org/XML/1999/XML-in-10-points, accessed September 25, 2003.

[XMLSpy2003] XMLSpy, http://www.xmlspy.com/, accessed August 7, 2003.

[XP2003] Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org, accessed September 11, 2003.

[Yeager2003] Yeager, W., "Cryptography Toolkit for JXTA Technology," http://security.jxta.org/Security-project.html, accessed September 26, 2003.

12. APPENDIX A: MASE P2P SITE

The tool can be downloaded from http://sern.ucalgary.ca/~bowen/research/masep2p/. The source code and help documentation is also available at the same site. To make use of MASE P2P, you need an Internet connection and be using a platform that supports a Sun Java Virtual Machine (JVM) [Java2003]. The tool can be set up and used with a minimum of resources because both MASE P2P and MASE can be deployed using proprietary-free applications (i.e., JBoss, Tomcat, and MySQL). There are two installation packages for MASE P2P: client and server. Some configuration is required before running, which is discussed in more detail in the online help file available at http://sern.ucalgary.ca/~bowen/masep2p/help.htm. The client package contains the GUI that allows the user to access and modify the P2P information for the MASE objects, and is intended as a lightweight installation that can be run on any machine with a JVM. The server package is intended to be installed on a machine that has MASE installed, which means having the following items installed: a JVM, a Web server, a J2EE application server, and a DBMS. The MASE P2P tool is being updated periodically, with the intention of removing bugs that are identified, and making changes to improve existing features. There are several features that would increase the usefulness of the tool, which are described in chapter 10 Future Work.

13. APPENDIX B: ENTERPRISE JAVA BEANS

The Enterprise Java Bean (EJB) technology is part of the J2EE (Java 2 Platform, Enterprise Edition) suite of technologies released by Sun Microsystems. J2EE is geared towards enterprise, or business, development and deployment [J2EE2003]. The EJB technology provides several benefits that were useful for the implementation of MASE P2P:

- *Mapping of objects to a relational database*. Objects can be mapped to a relational database, which means the developer does not need to implement JDBC connections to a database, and then parse text or bytes from the database into primitive data types or objects, respectively.
- *Transaction management when working with a database*. EJBs provide support for transaction management, which means that database concurrency problems, such as dirty reads and writes, can be prevented by making use of the appropriate transaction management mechanism.
- *Remote method invocation*. Methods in EJBs residing on one machine can be called remotely from software running on another machine. Remote Method Invocation (RMI) is also used when a program makes a call to a method in a program running in a different JVM on the same machine.

14. APPENDIX C: JXTA

14.1 Communication

JXTA makes use of XML for data exchange. XML is ideal for data exchange because it is an extensible and platform-independent language that supports internationalization and localization. XML is well supported by most IT organizations. For example, Sun, IBM, and Microsoft are all integrating XML into some of their technical strategies [Rein2002].

JXTA supports both reliable and unreliable communication. A reliable transaction is one in which a packet is guaranteed to arrive at the destination, while an unreliable transaction is one where there is no assurance that a packet of information will arrive at the destination. The reliable connections make use of TCP, which ensures that all packets are delivered. The disadvantage to using TCP is that there is more bandwidth overhead because additional packets are required for the transaction. Unreliable connections make use of UDP (User Datagram Protocol). Gnutella uses a TCP broadcast instead of UDP because UDP is generally more difficult to program around firewall configurations.

14.2 JXTA versus Other Technologies

At first glance, it may appear as though JXTA is very similar to some other P2P technologies, however there are important differences. The purpose of this section is to identify some of the differences between JXTA and the other related technologies to give some reasoning for why the JXTA technology was selected for the implementation of this system. These reasons are in addition to the aforementioned benefit of using JXTA because it is implementation in Java, which smoothes the integration with the Java MASE program.

- *Jini*. This is another technology developed by Sun Microsystems that supports communication between small devices (i.e., non-desktop portable devices) in an

ad-hoc network [Jini2003]. The major difference between Jini and JXTA is that Jini is only geared towards Java applications, while JXTA can be run by applications using any language, which makes JXTA a more flexible architecture.

- *Windows peer-to-peer networking*. Microsoft's initiative provides a SDK that includes a set of APIs for building P2P applications for computers running Windows XP. Thus, much like .NET, this technology is tied to the Windows technology. This is a more heavyweight technology that JXTA. JXTA was designed to be lightweight enough that applications using the JXTA technology could be run on small handheld devices, such as PDA (Personal Digital Assistant) devices and cell phones.
- *SOAP*. JXTA and SOAP (Simple Object Access Protocol) are quite different technologies. JXTA supports a fully functional P2P environment, with searching, membership, and other services that are useful for a P2P network. SOAP is completely different because it only defines the format for the exchange of structured and typed XML-based messages information between peers in a distributed environment.
- *Gnutella*. This is another open-source P2P technology. Gnutella is only designed to run over the HTTP protocol, which does allow it to tunnel through firewalls. Gnutella is not as flexible as JXTA because it cannot run directly over other network protocols, such as the TCP or UDP protocols.

14.3 JXTA Services

JXTA supports all services that are required in a P2P environment. The JXTA platform is defined by six protocols (Table 14-1). A peer can be lightweight because it only needs to implement those protocols that it requires. The project JXTA protocols do not require periodic messages of any kind or level to be sent within the network.

Table 14-1. The six JXTA protocols [Traversat2002].

Protocol	Description
Peer Discovery Protocol (PDP)	PDP allows a peer to discover other peer advertisements (peer, group, service & pipe).
Peer Resolver Protocol (PRP)	PRP allows a peer to send a search query to another peer.
Peer Information Protocol (PIP)	PIP allows a peer to learn about the status of another peer.
Peer Membership Protocol (PMP)	PMP allows a peer to join or leave a peer group.
Pipe Binding Protocol (PBP)	PBP allows a peer to bind a pipe endpoint to a physical peer.
Peer Endpoint Protocol (PEP)	PEP allows a peer to ask for routing information to route messages to another peer.

14.4 Searching

The JXTA search protocol defines a flexible and scalable mechanism. The JXTA Search discovery and access model uses a query routing protocol (QRP) for distributed information retrieval [Waterhouse2002]. Special peers are used as hubs. Hubs can be specialized by anything, such as geography, content similarity, or application. Queries are forwarded from hub to hub as necessary. JXTA searches depend on advertisements, which have a lifetime and so they will expire at a certain time. Searches can either proceed in a wide or deep manner. The centralized aspect of the protocol means queries are efficient, and it is simpler to implement security and membership policies [Botros2001].

15. APPENDIX D: GUI DEVELOPMENT

The GUIs were implemented using Java's Swing API, for which there are extensive tutorials available online [Swing2003]. Implementing GUIs can be time-consuming because of the amount of code required for even a simple interface. The NetBeans IDE (http://www.netbeans.org) was used to create the GUIs because it supports drag-and-drop Swing GUI development. The Eclipse IDE (http://www.eclipse.org) was used for the rest of the implementation. Whenever a change in a GUI was incorporated into the deployed code, two sections of code had to be copied from the NetBeans IDE and inserted into the Eclipse IDE.