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

Seth Bowen
Department of Computer Science
University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
bowen@cpsc.ucalgary.ca

Frank Maurer
Department of Computer Science
University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
(403) 220-3531
maurer@cpsc.ucalgary.ca

Abstract

Distributed software development support systems typically use a centralized client-server architecture. This approach has some drawbacks such as the participants may experience lengthy delays if they are located far from the central server, and the organization that runs the server must deal with the security and privacy issues that come with being in charge of a central repository of information. We are investigating whether this centralized control can be relaxed by using peer-to-peer (P2P) technology. In this paper, we address the motivations for using a P2P architecture, examine the design issues related to the development of a P2P software development support system, and discuss the implementation framework that will be used.

1. Introduction

Globally distributed software development has led to the creation of several applications that are in widespread use today. Examples include open source software, such as the Apache Web Server, the Linux operating system, the MySQL database, and the CVS (Concurrent Versions System). Distributed software development is typically conducted in a centralized distributed topology, which involves people working in isolation and then releasing their source code to a central repository. Any coordination and communication that occurs between the members is done mainly through message bulletin boards and email. There are industry products available that developers can use over a network (e.g., Microsoft Project, Rational XDE), and some tools developed in educational institutions (e.g., TUKAN [7] and CHIME [1]). However, most of these applications rely on a centralized architecture to aid in the collaboration process. As a result, users participating in a distributed project typically (a) need to rely on a central server to access information, (b) need to have a reliable and fast connection to the central server, and (c) need to trust the organization that runs the server.

In our research project, we are investigating whether this centralized control can be relaxed by using peer-to-peer (P2P) technology. We see the following advantages compared to centralized approaches:

- The reliability of the overall collaboration environment is increased because information can be easily replicated at several sites. As a side effect of the replication, there is the opportunity for performance improvements because information can be brought closer to users.
- Users have a tighter control over the information that is accessible by other developers. This is especially necessary in virtual software enterprises, where the need for information sharing in a project must be balanced with the needs of participating organizations on protecting their intellectual capital.

The remainder of this paper is outlined as follows. Section 2 contains a description of the client-server application, MILOS. Section 3 discusses a P2P-based extension of MILOS. Section 4 outlines the objectives of this research. Section 5 addresses the design issues for creating a P2P application. Section 6 describes how project development can be divided between peers. Section 7 includes details on the implementation framework that will be used, and finally section 8 covers our conclusions and the future work of this research.

2. MILOS

MILOS (Minimally Invasive Long-Term Organizational Support) is an open-source application that provides support for the coordination and collaboration of virtual software engineering teams [5]. We give an overview of the tool to provide a starting point for deciding what information can be distributed. MILOS includes the following developmental aids:

- a workflow engine supports the coordination of tasks and handling of work products in a project,
- an experience base stores process models for future use,
- a resource pool contains profiles of developers, and
- a metrics gathering utility.

The workflow engine supports the planning and execution of software development. The main data structures represent projects, processes, and variables. The processes and variables can be thought of as tasks and work products, respectively. A process can have zero or more input, output, and modify variables (which are variables that are modified during a process). A variable is an instance of a work product and it can only be the output for one process, however it can be the input to more than one process. Variable types serve as templates for variables, and so a variable is instantiated from a variable type. The workflow engine supports flexible development throughout the entire development life cycle because processes can be added, deleted or changed at any time.

The experience base includes process models. A process model is a hierarchy of processes that have associated inputs and outputs. The output from a process can be an input to a process at the same or different level in the hierarchy. The information assistant is associated with the experience base and the workflow engine. It can be used to search for information, such as tutorials based on a user's skill set, or according to the type of task assigned to a user.

The resource pool stores all agent and team profiles. The skills of agents can be queried when agents with specific skills are required.

The metrics utility generates metrics for the sizes of packages, classes, and methods based on the number of lines of code (LOC) and the complexity of the object. Measuring the size of work products implemented during development increments is important because this information can be used to improve future estimates of effort.

MILOS supports agile software development using some XP (eXtreme Programming) practices [6]. During the planning of a project, user stories can be created and managed. As well, pair programming can be coordinated and then carried out via Microsoft NetMeeting.

MILOS is an open-source application that can be deployed on several platforms without having to pay proprietary licences. MILOS is written in Java and makes use of Enterprise Java Beans (EJB) for mapping objects to a relational database. It is deployable using the following open-source applications: Apache Tomcat as the Web application server, JBoss as the J2EE application server, and MySQL as the relational database. Although MILOS is geared towards software development, the application is general enough to support the coordination of many types of project.

3. MILOS and P2P

MILOS is based on a client-server architecture that has some potential drawbacks for virtual software development teams. First, being dependent upon a central repository means that information may be brought together simply for the convenience of exchange or access. The potential problem is that the singular group in charge of this information must deal with the security and privacy issues related to this intellectual property, when several groups could perhaps more appropriately manage it. Second, a virtual environment often means that team members are distributed around the globe. The distance (not physical distance but the virtual distance that results from slow network connections) of a user to the central server can become a factor because team members may connect to the Internet via slow connections and so could be subjected to long delays.

The P2P networking model offers advantages in software development over a client-server model because of the independence from a central control. First, each peer, or peer group, is able to have complete control of its information, such as source code. The group could even be more cohesive because it can choose what information to filter from the outside. Second, groups can dynamically share and replicate information with one another to allow for faster access to users who may have slow network connections, or to support multiple projects that have dependencies on the same artifacts. Third, the developers can easily contribute their own resources, such as hard drive space and computing cycles, to an endeavour, which increases the robustness of the environment. This feature can be especially helpful for projects with limited funding, such as open-source projects.

There are also potential drawbacks that must be considered when adopting a P2P design. First, searching for information is in general more complicated and time-consuming in a P2P environment than in a client-server environment because the information is often more

distributed. This complexity could lead to longer development times. Second, there is the issue of the coordination of information in a P2P environment. For example, if information is replicated then how often should the stores be synchronized with one another? Furthermore, partitioning information requires understanding the dependencies between units of information. Third, the dynamic environment does not lend itself to peers being easily traced by static IP addresses or registered domain names. Fourth, the dynamic nature of the P2P environment means that information or services may suddenly become unavailable. Depending on the requirements of the organization, mechanisms may need to be implemented to deal with these issues.

4. Objectives

We will outline our research objectives in this section since we are still working on the first phase of our research. The objectives for our research of P2P virtual software development are threefold. First, to examine the design issues related to the development of a P2P-based virtual software development support system, with specific attention given to the coordination of data between different peers, and security concerns. Second, to alter MILOS from a client-server to a P2P application using a P2P framework. And third, to assess the value of the P2P implementation based on comparisons of the two implementations using criteria such as performance, and quality and quantity of information available. An empirical evaluation of the approach will involve gathering data on the development process, measuring performance by measuring the delays when retrieving information, and assessing the value of the altered system by user's impressions of whether the P2P implementation improved upon the original (e.g., with the use of questionnaires). With these objectives in mind, we hope to present some novel findings into the usefulness and issues related to applying the P2P architecture in virtual software teams.

5. P2P Design Issues

The first phase of this research is to think about the design issues when designing a P2P application. These issues are similar to those that should be considered when designing any distributed system. The design issues can be addressed with the following questions:

- 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?
- How can we find a balance between open information sharing and the protection of the intellectual property of members of the virtual team?
- What peer roles are defined?

We discuss these issues in the following sub-sections, and then outline how we addressed the issues in the design of our system.

5.1. What Information

All information not related to the execution of the MILOS application could be distributed among peers. However, we will concentrate on the main components of MILOS. Specifically, we will deal with the workflow engine objects (e.g., projects, processes, and variables), and the agent profiles in the resource pool. This range of information will enable us to examine how the P2P architecture can be applied to a variety of different situations.

5.2. Information Organization

Information is easier to understand and deal with if it is organized with related information. The unit size can be small or large depending upon the level of abstraction that is used to represent the information. Deciding whether to use smaller or larger units of information is dependent upon several factors, including the types of dependencies between information and the relatedness of the information. Transferring large units of information can be inefficient if there is a large amount of unnecessary data that is transferred with the desired data.

Synchronization overhead can result when using large units of information, because there is more information to synchronize at one time. On the other side, there may be more complexity when synchronizing small units because there will tend to be more dependencies between the units. No guideline will be given here for how to partition your information because the decision is highly dependent upon the domain.

Since MILOS is implemented in an object-oriented language, it is simplest to deal with information at some object level. Specifically, when considering flexibility in searching and dividing project tasks between peers, the following objects seem reasonable: projects, processes, variables, variable types, and agents.

5.3. Information Distribution

Two dimensions are used to define the way in which information is distributed: 1) the number of peers that can write the information, and 2) whether the information is replicated. Assuming there is no duplication of data, no mechanism to preserve data consistency is required if there is only one writer. However, if there is more than one writer than a locking mechanism is required to maintain data consistency.

Having duplicate information is useful for two reasons. First, reducing delays when information is accessed much in the same way that Web browsers use file caching. As well, the robustness of the system is enhanced because there are multiple copies of information.

Data synchronization is required when information is distributed over multiple locations. The information may be replicated, or information may have a dependency on information at a different location. It is important that it can be recognized when information requires updating, and there also has to be a mechanism to perform the update. Most operating systems keep track of when a file was last modified, but to keep track of when an object is last modified requires some implementation on the part of the programmer.

We want to make our system flexible, and so we will support multiple writers for units of information, as well as allow information to be replicated.

5.4. Searching

Search mechanisms allow users to discover new information. For user friendliness, and in the interests of flexibility, searching should be supported at several layers of abstraction.

Searching will be a major component of the new design because users will now be given access to a great deal more information than was previously available on one machine. For example, if a group requires a user with a specific skill then it can search for users outside of their group. Furthermore, an abundance of dynamic information is available due to the P2P environment. Users will have the ability to list information without having to enter search specific criteria. For example, a user could retrieve all versions of a specific variable.

Searching makes use of advertisements, which are specified in the JXTA protocols [8] as a means for advertising services and information. Advertisements are structured in XML (eXtensible Markup Language) and so are readable by programs irrespective of the type of platform. Users will be able to search for peer groups and peers within peer groups. As well, within a peer group, searches can be done for projects, tasks, and variables.

5.5. Protecting Intellectual Property

Like any situation where users are communicating with each other over a network, security is an important issue. Authentication is required so that a user can verify the identity of other users. Authorization measures ensure only the appropriate users have access to, and can manipulate, data. Encryption is used to ensure eavesdroppers cannot decipher private information.

Our system will be flexible enough to allow for information sharing while protecting the intellectual property of team members. Authentication will be enforced in the P2P system with an id and password. Authentication data will be encrypted when transferred. Other data will by default be encrypted unless the user wishes to send non-sensitive information unencrypted to reduce the amount of overhead in the data exchange. For authorization, the owner of a unit of information sets the access privileges for that information. The following access privileges will be supported: single owner, open for a set of peers to become owners, and readable by a set of peers. Note that a set could include no peers, or all other peers. If a peer has owner or read access to a unit of information, then the peer can decide if it wants to replicate the information.

Dealing with firewalls is another security issue because peers will often have to communicate with one another from behind firewalls. Firewalls prevent people from establishing unknown connections to systems, and allow information to be monitored on the few ports that are made available to the outside. In our system, peers will be able to communicate with one another from behind firewalls.

5.6. Roles

Roles are defined for peers and groups of peers. A peer group is composed of one or more peers. A peer group is useful because it supports a secure domain for the exchange of private information, and it can be used to create a scoping environment for peers with similar interests [8].

We define the following roles for peers: owner, helper, rendezvous peer, and relay peer. A unit of information can have more than one owner. The concept of ownership is used in order to help describe the type of data synchronization required. The creator of a unit of information can decide whether other peers can own the

information, and whether other peers or peer groups can read the information.

A helper is defined as a peer that has a duplicate copy of information. A mechanism is required for updating duplicate copies of information. Using a pull mechanism would require the helper knowing who owns the information. The update request could be periodic, or initiated when a peer tries to access the information. On the other hand, a push mechanism would involve the owner of the information initiating the update and knowing who the helpers are. In a P2P system, we feel that for simplicity, pulling information is a favourable approach over pushing because helpers may be offline when a push is done, in which case the owner would have to keep track of the helpers that are offline. Or, the helpers could pull information when they become online, which is a more complex solution because it uses a combination of the two approaches.

A rendezvous peer is essentially a knowledge hub (i.e., a central blackboard) for a peer group. Some of its duties would include storing advertisements for retrieval or distribution to other peers, and handling requests joining and leaving a group. These advertisements could include information about the group, such as data, services, or how to contact peers. The rendezvous peer is also appropriate for storing information centrally in a peer group, such as agent profiles, and authentication information.

A relay peer can be used for communication from behind a firewall. A peer could contact a rendezvous peer through a published common port, and then once the peer is authenticated an alternate connection could be established with the relay peer so that the common port remains available for general requests.

There are two peer group roles of interest: the MILOS group and developer groups. The MILOS group defines the group of peers that are using the MILOS system, which means peers within this group understand that they can collaborate on projects using MILOS data structures. Peers within the MILOS group can organize themselves into developer groups for specific projects, which allows them to distinguish themselves from other peers. A typical developer group may include an owner, a rendezvous peer, and a relay peer. Note that a helper for an owner does not necessarily have to be in the same developer group as the owner if a project is distributed over several developer groups.

6. Project Division

The P2P architecture facilitates the division of a project's processes between semi-autonomous groups. The groups are semi-autonomous in that the system will support the groups to be in control of their own development activities and information, while still being able to cooperate and contribute to an overall project.

Ownership is the key in the division of project tasks. In MILOS, a project plan is represented as a hierarchy with the project plan as the root, and processes and variables below. The peer that creates the project is the owner of the project plan. The owner of a project can divide up processes within the project by relinquishing ownership to all peers, or only to certain peers. This constraint is used so that no one becomes the owner of a process without actively choosing to become the owner. The owner of a process is also the owner of the process's output variables because variables can be inputs to processes owned by other peers. The owner of a variable that is the input to a process owned by another peer will be responsible for either informing the other peer, or synchronizing the updated variable if it changes. The owner of a process can choose whether its information is visible to other peers, and if it is visible then other peers can choose to copy the information.

Ownership propagates down the hierarchical structure. For example, any ownership or access privileges set for a process will apply to its sub-processes and variables. The owner can also alter the access privileges of these sub-processes if desired. Multiple owners are allowed for an object; however, if there are multiple owners then a locking mechanism is required.

7. Implementation Framework

We will be using the JXTA (pronounced "juxta") P2P framework for our implementation. It is a general specification for describing a P2P architecture because it does not restrict the type of language, service, or even network that is used [2]. For example, data communication can be implemented at the application layer by using HTTP, or even at a lower network layer by using datagram packets. The JXTA specification is intentionally high-level to support the interoperability of any P2P system. Data is structured in JXTA using XML, which is a common method of structuring data so that applications on any platform can understand the information. Thus, any application can be enhanced to take advantage of the JXTA specification without being tied into a vendor. Open-source implementations of the JXTA specification are currently being developed in several languages, including Java, J2ME (Java Micro Edition), C, Perl, and Python [4]. There is currently a stable build available in Java because it was the first programming language used to implement JXTA. Java is an appropriate language for the implementation of a P2P specification because applications written in Java can run in several different

operating systems, including Windows, Macintosh, Solaris, and Red Hat Linux [3].

The JXTA framework is composed of six protocols that cover the basic services required in a P2P environment, such as finding routes to peers, and sending queries and receiving responses [8]. A peer does not have to support all of the protocols, but rather only those protocols that it requires.

8. Conclusions and Future Work

A P2P architecture includes peers that are not constrained by capabilities or responsibilities. We propose that this type of architecture provides benefits for a distributed software development tool over the common client-server architecture. For instance, the flexible architecture supports cooperating groups to have greater control over their information, and the robustness of the network can be increased with little effort.

In this paper, we discussed some of the design issues for developing a P2P application, and outlined how we plan to adapt an existing application that is based on the client-server model. We are still in the process of clarifying the design details for the new system. Our goal is to make a flexible and robust system that takes advantage of the P2P architecture. We will implement the new system, and then compare the performances of the P2P version with the original to see if there is any unacceptable performance overhead. We also hope to evaluate the value of the P2P implementation by comparing the complexity of the two implementations (in terms of design), and user's perceptions of the added functionality. The thoughts presented in this paper on the design of a P2P collaborative software development tool provide a starting point for the future implementation, and analysis of the P2P approach.

9. References

[1] S.E. Dossick, D. Port, and G.E. Kaiser, "Embedding Model-Based Architecting in a Collaborative Environment", http://www.psl.cs.columbia.edu/ftp/psl/CUCS-016-99.pdf (current May 2002).

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

[3] Java 2 Standard Edition, v 1.4.0 Download, http://java.sun.com/j2se/1.4/download.html (current May 2002).

[4] JXTA Project, http://core.jxta.org/servlets/ProjectHome (current May 2002).

[5] F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kötting, and M. Schaaf, "Merging Project Planning and Web-Enabled Dynamic Workflow Technologies", *IEEE Internet Computing*, vol. 4, no.3, May/June 2000, pp. 65-74.

[6] F. Maurer and S. Martel, "Process Support for Distributed Extreme Programming Teams", http://sern.ucalgary.ca/~milos/papers/2002/MaurerMartel2002.pdf (current May 2002).

[7] T. Schümmer and J. Schümmer, "Support for Distributed Teams in eXtreme Programming", http://www.darmstadt.gmd.de/concert/people/schuemmePrivate/xp00.pdf (current May 2002).

[8] B. Traversat, M. Abdelaziz, M. Duigou, J.-C. Hugly, E. Pouyoul, and B. Yeager, "Project JXTA Virtual Network", http://www.jxta.org/project/www/docs/JXTAprotocols.pdf (current May 2002).