Online Cooperation Learning Environment

A thesis presented
In partial fulfillment of the requirements
For the degree of

Master of Science
in
Computer Science

At Massey University, Albany,
New Zealand

Jun Shang

2005

ABSTRACT

This project aims to create an online cooperation learning environment for students who study the same paper. Firstly, the whole class will be divided into several tutorial peer groups. One tutorial group includes five to seven students. The students can discuss with each other in the same study group, which is assigned by the lecturer. This is achieved via an online cooperation learning environment application (OCLE), which consists of a web based J2EE application and a peer to peer (P2P) java application, cooperative learning tool (CLT). It can reduce web server traffic significantly during online tutorial discussion time.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Chris Messom, for his guidance, patience and support during the period of this research project. Also, I would like to thank Dr. Abdolhossein Sarrafzadeh, for his guidance and support.

Finally, I would like to thank my wife Julia, for her endless support and encouragement.

TABLE AND CONTENTS

1 INTRODUCTION

Becoming better at learning is one of the most important goal for anyone participating in education or training programs. Cooperative learning is one of the solutions to help people to archive this goal. Bouton and Garth stated that learning is a group process: the learner actively constructs knowledge by formulating ideas into words and then the ideas/concepts are built upon through reaction and responses of others [22]. The cooperative learning occurs when a small group of students work together to maximize their learning capacity under the lecturer's instruction [2]. It is useful to help student to reappraise, if necessary, their thinking about how they learn best, so that they can take control of their learning processes consciously and develop them systematically. Cooperative learning has proven to be a successful method in traditional classroom settings [23].

Online Cooperation Learning Environment (OCLE) is an application that provides tool and solution for the cooperative learning by using Internet and the World Wide Web as a framework. Using such software, the learners can discuss and share files within their own group.

E-learning has become more and more popular in educational institutions. The E-learning means that learning is facilitated and supported through the use of information and communications technology; E-learning is to learn entirely online, from supported learning to blended learning [1]. E-learning presents a host of new opportunities for institutions to cost-effectively expand access to education and improve educational outcomes. The learner can get learning material and contact with another learner freely without any timing and location's limitation.

The existing commercial products of the Online Cooperation Learning Environment are part of an E-learning system. Also, there are many research projects, which implement the OCLE. The weaknesses of the existing OCLE application are either the feature of study group was vaguer or the client-server module was adopted.

In the past decade, the population of accessing internet was growing fast. People use internet to get information and communicate with each other. The communication tool, which based on internet, has been developed and proven successfully, such as Windows Live Messenger, ICQ and AOL Instant Messenger. Those applications are named as the instant messaging (IM). The instant messaging is a form of real-time communication between two or more people based on typed text. The text is conveyed via computers connected over a network such as the Internet. The framework of the IM makes the peer-peer computer based Cooperative learning possible.

1.1 Aim of my research

The aim of this research is to investigate and test the technologies by creating an online cooperative learning environment (OCLE) application. This application will conquer the shortcoming of server-client module by combining the P2P network and the web service. Also it tries to conquer the P2P module's shortcoming, which is loss of centralize control in order to meet cooperative learning's aims.

To achieve this goal, the whole application is divided into two components: OCLE controlling component, which uses JBOSS [7] application server to build a J2EE application and a cooperative learning tool component (CLT), which uses the P2P JXTA [6] network.

The OCLE controlling component, which is the J2EE application controls the group forming and monitors the group's performance. The lecturer inputs the group information and assigns student into a group.

The OCLE cooperative learning tool component is a windows application, which uses P2P JXTA network. Students can use this tool to chat and to share files within a study group. The cooperative learning tool will get group information

from the web service of OCLE controlling component in order to form a study group and upload group status to the web service of the OCLE controlling component.

The open source framework is adopted wildly in this application, such as the JBOSS application server and the P2P JXTA network. The open source framework is published and made available to the public which enables anyone to copy, modify and redistribute the source code without paying royalties or fees. The advantages of using open source framework are low cost and easily adjusting the framework by changing the framework's source code. After these frameworks have been changed, the OCLE cooperative leaning tool component and the OCLE controlling component can connect seamless.

1.2 Thesis Outline

This thesis is organized into the following 8 chapters:

Chapter 2 gives an overview of existing commercial applications and research projects as thesis background.

Chapter 3 gives an overview of the technology, which uses in this application and educational theory for Cooperative learning. The technologies include J2EE architecture, JBOSS application server, web service, and JXTA P2P network.

Chapter 4 gives an overview of the Online Corporation Learning Environment (OCLE) application and the two components' functionaries.

Chapter 5 gives the implementation details of the J2EE application and the P2P application.

Chapter 6 gives detailed testing procedures for the P2P application and the testing result.

Chapter 7 and 8 cover the shortcoming of this application and future analysis.

2 RESEARCH BACKGROUND

The concept of cooperative learning has been accepted by more educators. When the educator uses this concept in the traditional class setting, there are some limitations. For example, a physic location and a meeting time must be selected for a group of student and the lecturer must go around those groups in order to monitor the performance of the cooperative leaning. Recently, the internet technology has been selected by many educators to conquer these limitations. There are two types of internet products in cooperative leaning area: research projects and commercial products.

2.1 Existing Research Projects

2.1.1 First User Group of Cooperative Work

As early as 1980s, the first online user group of cooperative work appeared at the University of Essex [25]. It was created by the Richard Bartle and Roy Trubshaw and was named as the Multi-User Domain (MUD). It was initially confined to campus's student and then was extended to global user via modems. This was achieved by connecting the Essex University to the American ArpaNet, the precursor of the Internet.

The group of student used the MUDs to work together along with the FTP and email. The MUDs became popular with college students. By 1984, there were more than 100 active MUDs and variants all around the world.

The MUDs is pioneers of new forms of human interaction via the internet. On late time, many researchers and programmers of cooperative work get the idea from the MUDs and develop a new framework and application.

2.1.2 Teach++ project

Teach++ project [26] was created by the Di Salerno University in Italy. The Teach++ project is a cooperative environment, which specializes for the distance learning. Using Teach++ project, the teacher can give students tutorial online, assign the working group and upload the study material; the student can collaborately work on any project development online within a working group and retrieve the study material.

2.1.2.1 Architecture of Teach++

The client-server architecture and Java language are adopted by the Teach++ project. A Multi-Applet Server (MAS) is used to collaborate in the distributed computation on server's side. The MAS keeps the connection for any applet of a single client and manages communications among clients. The Java applet on the client establishes an independent connection with MAS.

The ClientChat and ClientTeach components operate on Client of the Teach++. These components are implemented by Java applets embedded in a multi-frame HTML page. The ClientChat component provides a synchronous communication channel among group members. The teacher and the student can use the ClientChat to send message to each other.

2.1.2.2 Teach++ Appraisal

In order to provide rich user interface, the applet is adopted by Teach++ project. The limitation of the applet, which they cannot perform connections to other hosts but the one they were downloaded from [27], forces the Teach++ project to adopt the server-client architecture. Every message, which the client is sending, must go through the server. The high performance server machine must be installed for the Teach++ application and the network traffic will be increased.

The Java applet was concerned by the OCLE application as the client interface for student, but it was abandoned later due to the disadvantage of server-client communication module. In order to conquer those disadvantages, the P2P communication module was adopted by my project.

2.1.3 WebDAV-Based Collaborative Distance Learning (CDL) Environment

The WebDAV-Based Collaborative Distance Learning Environment [28] is a web based application, which supports collaborative learning among a small group of learners amongst distance learners in a virtual university.

2.1.3.1 CDL Architecture

In order to directly address the complex computing requirements, Java 2 Platform Enterprise Edition (J2EE) is adopted as the foundation of the CDL environment. The EJBs, servlet and jsp technology are used to implement the CDL application. The CDL application uses the IBM HTTP Server 1.3.6.2 as a web server and the WAS AE as the EJB container.

The collaboration-friendly internet communication protocol: Web-based Distributed Authoring and Versioning (WebDAV) [29] are used by the CDL environment as the communication protocol rather than using HyperText Transfer Protocol (HTTP) protocol. The WebDAV is a groupware protocol, which extends the HTTP/1.1, provides a coherent set of new methods, headers, XML- based request and response entity body formats to directly support collaborative work on the web.

The IBM DAV4J 1.0.34, which consists of a client-side API and a server-side servlet to provide full support for class-2 WebDAV, implements the WebDAV's functionality. The application server and WebDAV framework, which are used in the CDL environment, are commercial products.

2.1.3.2 CDL environment Appraisal

In order to support web based collaboration, the CDL environments combines the J2EE platefrom and the WebDAV protocol together. Using the J2EE platform, the complex, trivial and repeated low-level development work are avoided. The collaboration-friendly WebDAV protocol provides full support to web based collaboration activities. This approach is adopted by the OCLE application. In OCLE application, the J2EE platform is adopted. Instead of using the WebDAV protocol, the OCLE application uses the JXTA framework to support cooperative work.

Apart from shortcoming of the server-client architecture, the commercial products are used in CDL environment, which includes the IBM J2EE application server and the IBM's WebDAV implementation. The commercial products provide stable implementation and support, but the developing cost has been increased significantly. Because the developing cost is important factor for the most research project, instead of using commercial product, the open source product is adopted by the OCLE application.

2.1.4 Rowan Virtual Meeting (RVM) System

The Rowan Virtual Meeting System [30] is a synchronous, on line web audio/video conferencing application, which has been developed by the RVM project team of the Rowan University. Using the RVM system, the student can collaborate online with students and lecturers in real time, between either individuals or large groups—with just a PC and an Internet connection.

In order to implement virtual meeting in real time, the Flash is adopted as a basic media type and the Macromedia Flash Communications Server is used as the server side framework, which manages the Flash communication and provides

access control. The Flash Player software is needed on the client computer to run Flash communications application.

The Flash is good media type for real time audio/video meeting because the size of the flash file is much smaller than the traditional media type. The communication of the RVM still sticks on the server – client which leads to the high network traffic.

The online web audio/video conference and the Flash communication can be combined into the OCLE application in the further. The communication model should be changed into P2P JXTA in order to reduce the network traffic.

2.2 Existing Commercial E-learning Systems

The existing commercial products of Online Cooperation Learning Environment are part of an E-learning system. Existing products, which contain OCLE, include WebCT [3] and LAMS [4].

2.2.1 WebCT

WebCT, Inc provides the most popular E-learning System in the world. Thousands of institutions in more than 70 countries are using this System. WebCT Campus Edition 6 is the newest version of WebCT 's E-learning System. It provides easy course preparation, efficient course management, and innovative teaching and learning tools. The WebCT Campus Edition 6 has Who's Online tool [5] for Cooperation Learning Environment (OCLE). Figure 2.2.1 shows the user interface for Who's Online tool.

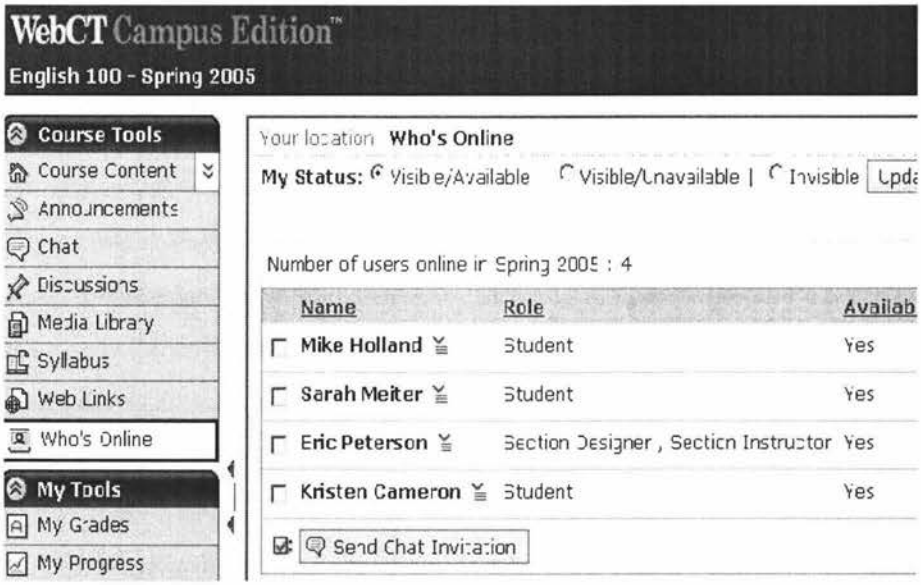


Figure 2.2.1 WebCT Campus Edition 6 -Who's Online tool

According to the paper and which student has been enrolled, the student can access learning material, which the lecturer distributed, and the student also can use the who's online tool to find and chat to another student who is studying the same paper.

However, there are no file sharing and peer group functionality in the who's online tool. The lecturer has the ability to mute or deny access to manage participation. Strictly speaking, the who's online tool of the WebCT is not an application of OCLE.

The who's online tool is a client-server network application. There is a chat server module hosting in WebCT application server to handle all conversations between learners.

2.2.2 LAMS

Learning Activity Management System (LAMS) is a revolutionary tool for designing, managing and delivering online collaborative learning activities [4].

LAMS tool focuses on sequencing leaning activities. Using LAMS, the educator assigns a task to a group of students. The group of students is predetermined by the educator. Students within the same group can do their task cooperatively by using the chat and sharing tool. Figure 2.2.2 shows the interface, in which the educator can edit learning content and group members.

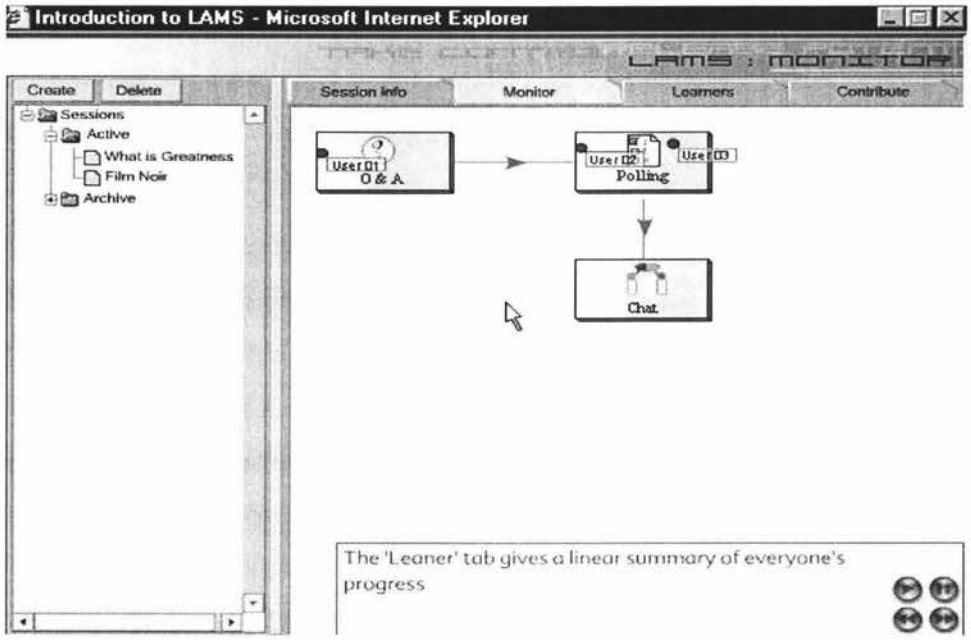


Figure 2.2.2 LAMS's learning content editor for lecture

LAMS is based on the java programming language and it allows the application to have a clean separation of presentation, logic and workflow. The LAMS server handles chat session between users.

2.2.3 Existing Commercial System Appraisal

Both of LAMS and WebCT are client-service applications. Client-server OCLE applications have some advantages; centralization efficiently handles participant coordination, since all names and IP addresses are kept in the main server; using user friendly web browser ' user interface which doesn't need to be installed; As part of the application, it can combine application server likes JBOSS and

Weblogic [21] etc easily. Despite its advantages, centralized OCLE applications need to use a high quality web server and large bandwidth to handle network traffic, since every message or file, which the learner sends to each other must go through the web server. The problem will become more serious, when multimedia is more popular during online discuss sessions.

The existing commercial OCLE products are expansive. The Educator that plan on having a subscription to use the full version of the WebCT campus edition can expect to make an annual payment upwards of $15000 or more, based on the number of students enrolled at the college [31]. This is big disadvantage for the most educators.

2.3 Valuable Insight

Both commercial and research products have a common shortcoming, which is using client – server architecture. This shortcoming can be addressed by building the application using peer-to-peer concepts. Any two computers directly exchanging resources have a peer-to-peer connection. The essence of peer-to-peer is that resources and services are in every node of the network and not to gather in servers. This model has gained popularity during the last few years because of an increase in computing power, computer storage space, and network bandwidth. In order to implement P2P communication in this project, the JXTA P2P framework was adopted as the basic communication's architecture.

Concerning cost efficiency, there are not any commercial framework and application server used in this project. All framework and application server are open source products, which mean you can free use those products and edit the source code easily. It reduces developing cost significantly.

The OCLE application adopted the CDL environment's architecture, which uses the J2EE as foundation and use third party framework as protocol of client's communication.

In order to provide a rich client interface, the Java applet technology is still used as the interface for teacher's administrator tool of the OCLE application. Instead of using the method, which the applet connects to the MAS server in the Teach++ project, the applet connects to the servlet container of the J2EE application server in my project.

3 BASIC CONCEPTS

This chapter provides an introduction to Project JXTA, J2EE, Jboss application server, web service and Jboss.NET. These concepts provide understanding of this project and will be applied to this project.

3.1 J2EE and JBOSS

3.1.1 J2EE

The Java 2 Platform Enterprise Edition (J2EE) is a complete development platform, which uses Java for server programming and provides a component-based approach to the design, development, assembly, and deployment of enterprise applications. The J2EE platform offers a multitiered distributed application model, reusable components, and a unified security model, flexible transaction control.

3.1.1.1 Container

A J2EE container is a runtime environment, which manages application components, and provides access to the J2EE APIs [7]. Basically, there are two types of container used in this project:

- A web container for hosting java servlets and jsp pages.
- An EJB container for hosting Enterprise JavaBean components.

Low-level semantics of enterprise development such as connection pooling and event transaction are built into a container-based architecture. Only the high level development needs to be considered in the program.

3.1.1.2 Component

The reusable component is an application level reusable unit. The J2EE applications are made up of components. The following components have been defined by the J2EE specification:

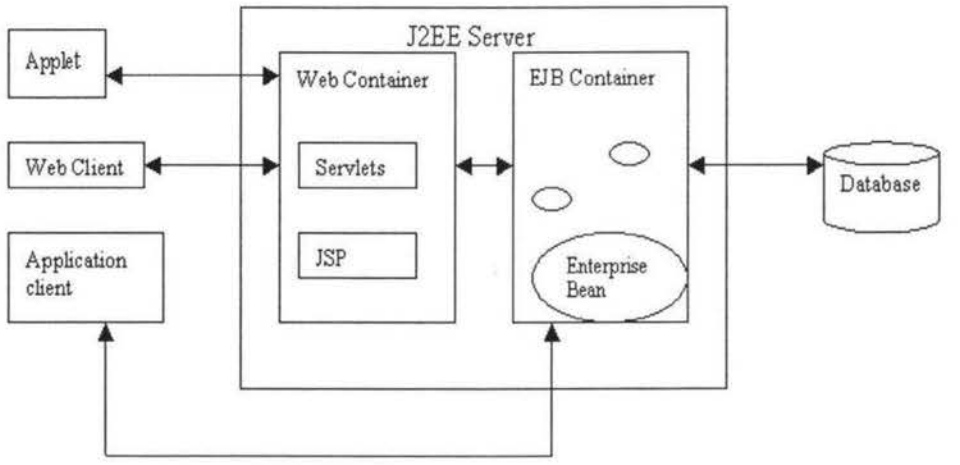
1. Application clients and applets are components that run on the client. An applet is a small client application written in the java programming language that executes in the Java virtual machine installed in the Web browser. The applet offers a sophisticated enough user interface (UI), which a standard HTML view cannot offer.
2. Java Servlet and JavaServer Pages™ (JSP™) technology components are Web components that run on the server.
3. Enterprise JavaBeans™ (EJB™) components (enterprise beans) are business components that run on the server.

The EJB architecture is a distributed component model for developing secure, scalable, transactional, and multi-user components. They are reusable software units containing business logic. EJB has two types of beans as follows:

a. **Session Bean.** A Session bean is an enterprise bean that represents work performed for a client. The session bean has two types. **Stateful session bean:** a stateful session bean is a transient object to represent a client's interaction with the system. It holds a state. **Stateless session bean:** maintains no state between client requests.

b. **Entity Bean.** An Entity Bean is a persistent object that models the data held within the data store that is an object wrapper for the data.

The various application components are installed on different tiers in the multitiered J2EE environment. Figure 3.1.1.2 shows the J2EE application architecture



.

Figure 3.1.1.2 J2EE architecture

All components, which were described here will be used in this project. This is given in detail in chapter 5.

3.1.2 Jboss application server

Jboss[6] application server is a free J2EE based application server, which is the most widely used Open Source application server on the market. It provides the full Open Source J2EE stack.

Jboss provides Jboss Server, the basic EJB container, and Java Management Extension (JMX) infrastructure. The JMX is a backbone of Jboss application server. The Jboss uses the JMX to integrate the software. The JMX provides a common spine that allows integrating modules, containers and plug-ins.

An abstract integration layer of the Jboss provides support for web components, such as servlets and JSP pages. Implementations of the integration service are provided for third party servlet engines like Tomcat and Jetty.

The Jboss version 3.2.5 was chosen as application server for this project. This project uses a tomcat 5.0 as web server, which is default web server for Jboss version 3.2.5 and replaces the default HSQL database with MySQL 4.0.

3.2 Service – Oriented Architecture

The Service- Oriented Architecture (SOA) is an instance of the Composite computing module. The composite computing module is an architecture that uses distributed, discovery-based execution to expose and to manage a collection of service-oriented software assets [9].

In an SOA, the capabilities should be dynamically discoverable. There should be a clear separation of the software's capabilities and its implementation and it should be possible to quickly assemble impromptu computing communities with minimal coordinated planning efforts, installation technicalities or human intervention. Figure 3.2 shows the SOA architecture.

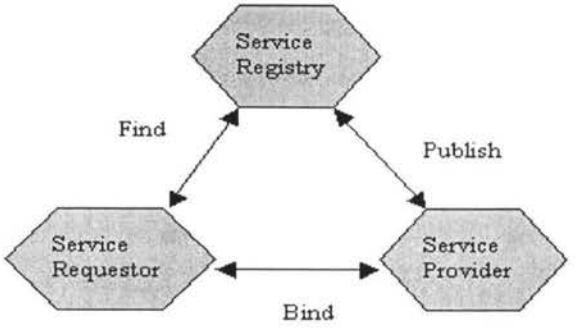


Figure3.2 Service-oriented architecture

The SOA concept is used through out the whole project. The OCLE controlling component's web service module and the cooperative learning tool component's chat server module and chat client module use the SOA. The details of those three modules will be introduced in the implementation chapter 4.

3.3 Web Service and JBOSS.NET module

3.3.1 Web service

Web service is a distributed systems technology that uses standard Internet protocols to move XML documents between service processes [8]. It is therefore web service is a software program that enables applications to talk to each other remotely via XML messages.

Briefly, a program sends a request to a remote Web service containing an XML message. It receives a response, which specifics how such services are represented, advertised, discovered and communicated. These responses are all defined by Web service standards.

Using Web Service to build an SOA, it is the ideal approach. Using standardized XML technologies in the Web Service, a service can be described, discovered and invoked. There are three main components of the Web service technology involving:

1. Simple Object Access Protocol (SOAP)

SOAP is a "protocol that can be used to exchange structured information in a decentralized, distributed environment" [10]. It is an envelope for a web service that contains the XML message.

2. Web Services Description Language (WSDL)

WSDL is "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedural-oriented information" [11].

3. Universal description, discovery and integration (UDDI) [12]. The UDDI is the yellow pages (repository) for a number of deployed Web service.

SOAP messages conform to the WSDL definition of available Web services.

Typically, WSDL defines the interface and location of the Web service, the SOAP message used to access the Web services and the protocols over which such SOAP messages can be exchanged and the WSDL descriptors can be accessed via a UDDI repository. Figure 3.3 shows the processing of this operation.

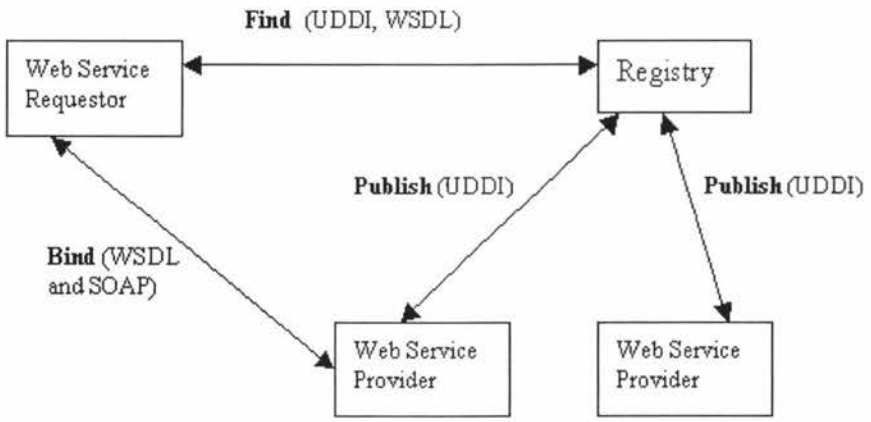


Figure 3.3 An overview of the SOA with web service protocol

The illustration of Figure 3.3 is as follows:

The Web Service provider creates a WSDL interface for its Web service that specifies its functionality, location of the service and the transport mechanism. Therefore, the Web Service provider publishes the service to registry (register the WSDL document with UDDI server). When a web service requestor wishes to use a service, it contacts the UDDI server, and gets location of WSDL file (normally,

it is URL address) and uses the WSDL file to create a request/response invocation on the web service. The message of the invocation is wrapped in an envelope, using SOAP, and sent to Web service.

Because a UDDI server is not free to use, the web service component of this project was simplified. After the WSDL file was located into an URL, the cooperative learning tool gets the WSDL file from URL, which has been built into the cooperative learning tool's application directly.

3.3.2 JBoss.NET module

The Jboss.NET is an early Web service, which was installed in Jboss 3.x series and was not installed in Jboss 4.x series. JbossWs replaced the Jboss.Net in Jboss 4.x series. Support for web services is provided via Apache Axis version 1.1. Axis [13] is used to handle the SOAP details and Jboss provides custom deployers and integration. Additionally, a JBoss.NET tag library is provided for XDoclet to make integration smoother. A standard JBoss.NET web service consists of a stateless session bean and a set of metadata describing its web service properties.

Using Jboss.NET, the EJBs of J2EE exposes as web services that are accessible via the standard SOAP 1.1 over HTTP protocol for use by Java and non-Java clients. This project uses this feature to implement communicating between the OCLE controlling component and the OCLE cooperative learning tool.

3.4 JXTA P2P Network

Interoperability and platform independence are missing attributes on today's peer-to-peer application. Each of those applications has their own protocol and frame, such as MSN Messenger, AOL Messenger and ICQ, are proprietary and incompatible with each other. Each application can only form its own private network outside of which it cannot easily exchange messages.

If a P2P application is needed to be implemented, the developers need to refocus heir efforts from programming P2P network fundamentals to create P2P applications on a solid, well-defined base. To do this, the P2P developers need a common language to allow peers to communicate and perform the fundamentals of P2P networking.

The purpose of project JXTA (pronounced juxtapose or juxta) is to create a platform with enough standardized functionality to enable open-source and commercial developers to create interoperable services and applications [15]. It originally formed by Sun Microsystems under the guidance of Bill Joy and Mike Clary. As the JXTA's pronunciation –juxtapose (in side by side), the project JXTA is a different communication module from today's popular client-server communication module.

The core of the JXTA is a set of protocol specifications and an open source reference implementation for developing peer-to- peer applications. The protocol of the JXTA can be implemented in any programming language, and run on any system platform, on any device, over any network protocol. It meets the requirement Interoperability and platform independence.

The developer can use the JXTA's real-time library supporting peer discovery and communication to create a P2P application. This is why the JXTA was adopted as basic architecture of the OCLE cooperative learning tool component. The P2P network fundamental of the cooperative learning tool component is implemented using JXTA.

3.4.1 Overview of JXTA Architecture

The architecture of the JXTA specification is divided into three layers, as shown in Figure 3.4.1.

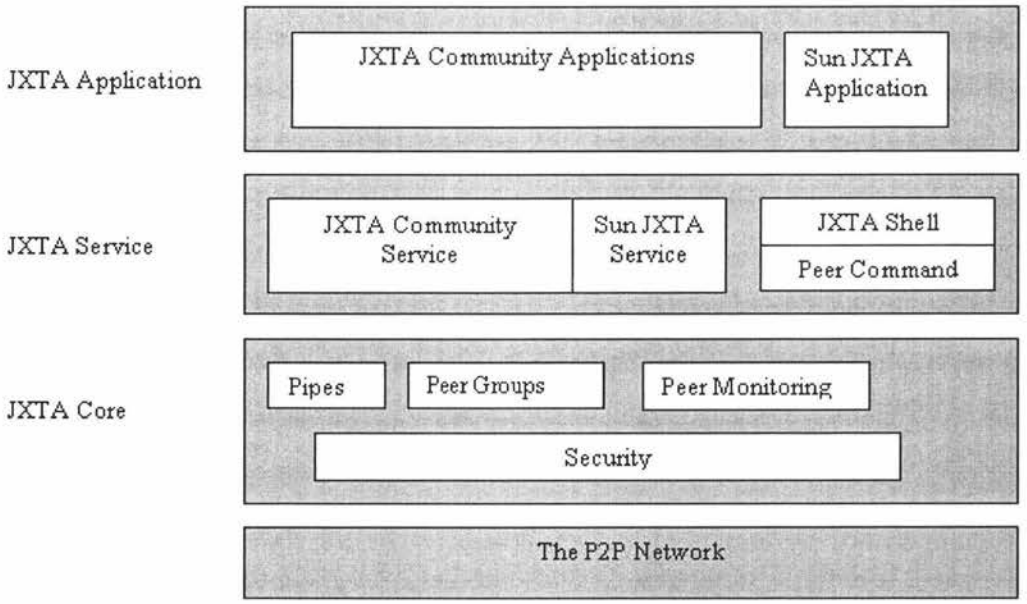


Figure 3.4.1 Project JXTA software architecture [14].

The Core Layer (platform layer)

The code for implementation of the JXTA's protocol is put in the core layer. The core layer is a core of the JXTA and includes building blocks to enable key mechanisms for P2P applications, including discovery, transport (including firewall handling), the creation of peers and peer groups, and associated security primitives.

Sitting over the protocol is a universal peer group called the **WorldPeerGroup**. When a peer starts executing, it will automatically become part of the WorldPeerGroup, and will access the service, which has been implemented.

Services Layer

The services layer, which built on top of the core layer, includes network services for a P2P network to operate. It includes searching and indexing, directory,

storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation and authentication services.

Applications Layer

The applications layer includes implementation of integrated applications, which is used by developers and pulls individual peers tighter for a common piece of functionality-for instance, to decipher an encrypted code.

3.4.2 JXTA Concepts

From section 3.4.2.1 to section 3.4.2.9 introduces the basic terminology of JXTA and their functions.

3.4.2.1 Peer

"A peer is any networked device that implements one or more of the JXTA protocols" [14]. The peer is a node in the JXTA network that can be an application distributed over several PCs or a device such as PDA or Mobile that connects to internet or even a single PC running multiple peer instances. The peer is uniquely addressed by their JXTA ID, satisfying the requirement for peer identification. The peer ID uses a JXTA's **identifier**, which uses UUID, a 128-bit datum to refer to an entity. The peer ID is a logical identification and is not a physical network point. When IP address of a peer is changed, the peer ID does not change. There are 3 types of peer in JXTA specification.

Edge Peer: A simple peer can send and receive messages, and will typically cache advertisements, but does not forward any discovery requests. Peers outside the firewall will probably not be capable of directly communicating with the simple peer located inside the firewall.

Rendezvous Peer: Due to absence of a central service (such as domain name server), a JXTA network uses a rendezvous peers. The rendezvous peers are volunteers that have agreed to act as a caching server for other peers. Rendezvous peers often maintain a permanent IP address, so that other peers can contact them to check the current bindings of dynamic peer endpoints. Each rendezvous peer can augment its capabilities by caching information on peers for future use or by forwarding discovery requests to other rendezvous peers. The rendezvous peer will usually exist outside a private internal network's firewall. If a rendezvous peer exists behind a firewall, it needs a relay peer or a protocol authorized to contact the peer out of the firewall.

Relay Peer: A replay peer maintains information about the routes to other peers and routes messages to peers. A peer first looks in its local cache for route information. If its not found, the peer sends queries to relay peers asking for route information. Relay peers also forward messages on the behalf of peers that cannot directly address another peer (e.g., NAT environments), bridging different physical and/or logical networks.

Figure 3.4.2.1 shows a scenario of how peers can communicate between each other.

JXTA Peers B and D want communicate with each other, but the firewall prevents them from communicating directly. The peer B is an edge peer, and is configured to use Peer R1 as its rendezvous and the peer D also is an edge peer, and is configured to use Peer R3 as its rendezvous and relay peer. Now, the peer B sends a query request for the peer D's location to its rendezvous (R1). The rendezvous R1 will check its local cache and will propagate the query to R3 if it is not found. The R3 find location of D in its local cache and send it back to the peer B. JXTA Peer B first makes a connection to the R3 using a protocol such as HTTP that can penetrate the firewall. The R3 then makes a connection to the peer D, using a

protocol such as TCP/IP. A virtual connection is now made between Peers B and D.

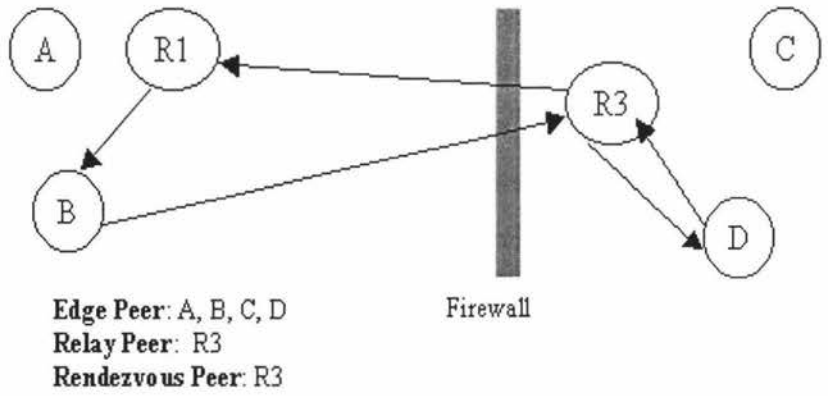


Figure 3.4.2.1 Peer's communication

3.4.2.2 Peer Group

"A peer group is a collection of peers that have agreed upon a common set of services" [14]. Peers in the Project JXTA network self-organize into peer groups. A peer group represents a dynamic set of peers that have a common set of interests, and have agreed upon a common set of policies (membership, content exchange, etc.). Each peer group is uniquely identified by a unique peer group ID. Project JXTA does not dictate when, where, or why peer groups are created. Project JXTA only describes how a peer group is created, published, and discovered. Users, service developers, and network administrators dynamically create peer groups to scope interaction between peers, and to match their applications demands. A peer can belong to multiple peer groups.

At boot time, every peer joins the **NetPeerGroup**. The NetPeerGroup acts as a global peer group to which every peer initially belongs.

3.4.2.3 Advertisements

An advertisement is an XML structured document that names, describes and published the existence of a resource. The JXTA defines a basic set of advertisements basic types Using XML schemas, or more typically by using the particular language binding like Java (Advertisement class).

The JXTA standardizes the following advertisements:

- Peer advertisement
- Peer Group advertisement
- Pipe advertisement
- Module advertisement
- Peer Endpoint advertisement.

Figure 3.4.2.3 shows an example of a peer Group Advertisement. It consists of a peer group ID, a module specification ID (MSID) that points to a module Implement advertisement, which describes all of the peer group services in this peer group, a group name (Name), and a peer group's description (Desc).

```
<?xml version="1.0"?> <!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
  <GID>
   urn:jxta:jxta-NetGroup
  </GID>
  <MSID>
      urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206
   <MSID>
    <Name>
      NetPeerGroup
    </Name>
    <Desc>
       NetPeerGroup by default
    </Desc>
</jxta:PGA>
```

Figure 3.4.2.3 An example of the peer group advertisement

3.4.2.4 JXTA Pipe

Pipes are virtual communication channels used to send and receive messages between services and applications. Pipes provide a virtual abstraction over the peer endpoints to provide the illusion of virtual in and out mailboxes that are not physically bound to a specific peer location. Pipes can connect one or more peer endpoints. At each endpoint, software to send, receive, or manage message queues or streams is assumed. The pipe ends are referred as the input pipe (receiving end) and the output pipe (sending end). When a message is sent into a pipe, the message is sent by the local output pipe to the destination pipe input which currently listening to this pipe.

The pipes offer two modes of communication, point to point and propagate as seen in Figure 3.4.2.4.

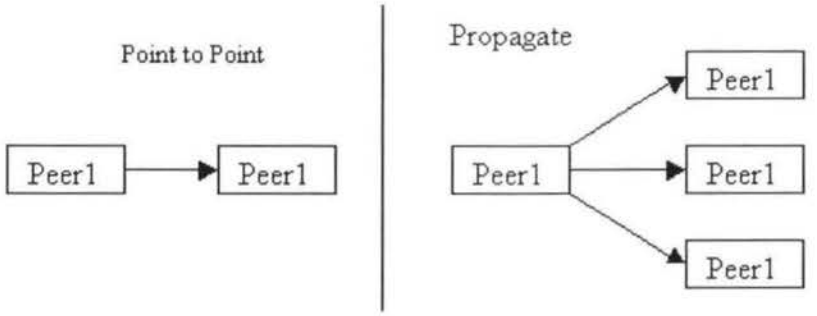


Figure 3.4.2.4 Pipe's communication modes

The pipes are published and discovered using Pipe Advertisements, and are uniquely identified by a Pipe ID.

3.4.2.5 JXTA Protocols

"JXTA defines a series of XML message formats, or protocols, for communication between peers" [14]. There are six protocols in the JXTA specification.

- *The Peer Discovery Protocol (PDP)* A peer uses the peer discovery protocol to discover a JXTA resource.

- *Peer Resolver Protocol (PRP)* The PRP enables a peer to implement high-level search capabilities, allowing a peer to send and receive generic queries to find or search for peers, or other genetic advertisements.

- *Peer Information Protocol (PRP)* The PRP allows peers to learn about the capabilities and status of other peers.

- *Endpoint Routing Protocol (ERP)* The ERP allows a peer to find information about the available routes for sending a message to the

destination peer.

- *Pipe Binding Protocol (PBP)* The PBP allows a peer to establish a virtual communication channel between one or more peers.

- Rendezvous Protocol (RVP) The RVP allows a peer to send messages to all the listeners of the service.

A peer can choose to implement the protocols it wishes and then rely on other peers to provide them with extra functionality. Those protocols are not totally independent of each other. Each layer in its protocol stack relies on it layer below to provide connectivity to other peers (see Figure 3.4.2.5).

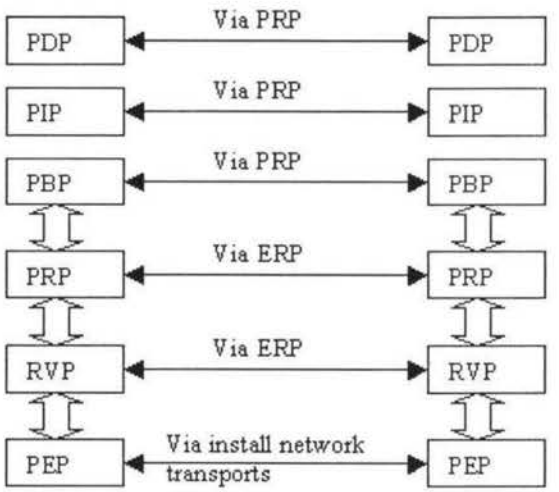


Figure 3.4.2.5 the layering of the six JXTA protocols

3.4.2.6 JXTA Service

The JXTA Services provide functionality that peers can engage to perform any task including file transfer, status information, sending a message or even asking

for the current time on a remote peer. The JXTA Services can be divided in two different levels:

Peer Service: Service was offered by a peer, which is on the network, to another peers. If the peer, which has been provided the service, was off the line, the service becomes unavailable.

Peer group Service: A service was provided by a peer group. The member of this peer group can use this service. "A peer group service is composed of a collection of instances (potentially cooperating with each other) of the service running on multiple members of the peer group" [14]. If any peer of this group is offline from the network, the peer group service still available until the last member is disconnected to the peer group.

A peer has two ways to get a peer group service, pro-install or download it from the network. It is similar to install a plug-in for a web page. This is the reason why every peer become member of the *NetPeerGroup* at boot time. The NetPeerGroup has Discovery Service, Pipe Service, Rendezvous service and Peer Resolver service, so after becomes member of the NetPeerGroup, every peer has the network's abilities.

3.4.2.7 JXTA Modules

"Jxta modules are an abstraction used to represent any piece of 'code' used to implement a behaviors in a JXTA network" [14]. The module abstraction does not specify what is this "code" is: it can be a Java class, a Java jar, a set of XML messages, or a script. The implementation of the module behavior is left to module implementers. The module framework is designed to allow a developer to provide functionality within JXTA in an extensible manner. Modules managed by the framework are responsible for providing all aspects of JXTA's functionality, including the implementation of the *peer group service* that are provided by a

peer group. The modules is a bridge, which a peer can load a service within a group. The new peer group service can be used in a peer group through the module framework. The module framework in the JXTA is the Service - Oriented Architecture (SOA) and are similar to the Web service. The advertisement of module is like the WSDL file in Web service.

To enable peers to discover modules, the definition of a module is divided into three types of advertisements: a Module Class Advertisement, a Module Specification Advertisement, and a Module Implementation Advertisement.

The Module Class Advertisement

The first advertisement, the Module Class Advertisement, doesn't provide information on a module implementation; it exists solely to announce the existence of a class of module. The Module Class Advertisement consists of following information:

MCID—A required element contains a Module Class ID. It is contained in the Module Specification and Implementation Advertisements.

Name—An optional element contains a simple name for the module class. This string is not necessarily unique.

Desc—An optional element contains a description of the module class.
Each class of modules has a unique Module Class Advertisement.

The Module Specification Advertisement

The Module Specification Advertisement is responsible for defining a module. The purpose of a Module Specification Advertisement is to uniquely identify a set of protocol compatible modules.

The following elements consist of the module specification advertisement.

MSID- Each module specification advertisement is identified by a unique ID, the ModuleSpecID. The ModuleSpecID contains the ModuleClass ID, indicating the associated module class.

Name—An optional element contains a simple name for the module specification. This string is not necessarily unique.

Vers—A required element contains information on the specification version is embodied by this Module Specification.

Desc—An optional element contains a description of the module specification.

Parm—An optional element contains parameters for the specification. The format and meaning of these parameters are defined by the module's specification.

JXTA:PipeAdvertisement—An optional element contains a Pipe Advertisement describing a pipe that can be used to send data to the module. This element is actually the root element of the Pipe Advertisement, not an element that contains a Pipe Advertisement.

For every Module Class Advertisement, there can be one or more different Module Specification Advertisements, each specifying a different version of the module.

The Module Implementation Advertisement

The Module Implementation Advertisement provides information on a concrete implementation of a module specification:

MSID—A required element contains the Module Specification ID identifying the module specification that this module is implementing.

Comp—A required element contains compatibility information. The format of the information icontained by this element depends on the possible deployment platforms for modules.

Code—A required element contains any information requires to run the code of the module implementation.

PURI—An optional element contains a URI, which points to a package that contains the code responsible for providing the module implementation. If the peer application cannot find code in pro-install package, it will downloads from URL which in define in this element.

Parm—An optional element contain parameters for the implementation.
The format and meaning of these parameters is defined by the module's implementation.

For every Module Specification Advertisement, there can be one or more different Module Implementation Advertisements, each specifying a different version of the module. Modules that are described by different module implementations that point to the same Module Specification Advertisement are compatible. Figure 3.3.2.7 shows an example of the Module Implementation Advertisement

```
<?xml version="1.0"?>
<!DOCTYPE jxta:MIA>
<jxta:MIA xmlns:jxta="http://jxta.org">
    <MSID>
        urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306
    </MSID>
    <Comp>
        <Efmt>
            JDK1.4
        </Efmt>
        <Bind>
            V1.0 Ref Impl
        </Bind>
    </Comp>
    <Code>
        net.jxta.impl.peergroup.StdPeerGroup
    </Code>
    <PURI>
        http://www.jxta.org/download/jxta.jar
    </PURI>
    <Desc>
        General Purpose Peer Group Implementation
    </Desc>
    <Parm>
    </Parm>
</jxta:MIA>
```

Figure 3.4.2.7 An example of Module Implement Advertisement

The Modules framework is a main frame for the OCLE cooperative learning tool component of this project. New peer group services are developed into this component and the student uses those new services to cooperate learning within a group.

3.4.2.8 Peer's life Cycle

Figure 3.4.2.9 shows the peer's life cycle. The first thing that the JXTA platform requires when bootstrapping is a World Peer Group, which is a peer group identified by a special Peer Group ID. The World Peer Group defines the basic capabilities of the peer, such as the services, endpoint protocol implementations, and applications that the peer will make available on the network.

After the World Peer Group has been created, the peer needs to instantiate the

Net Peer Group. The Net Peer Group can describe additional characteristics about a peer, but most often it is simply a duplicate of the World Peer Group. After the Net Peer Group is instantiated, the services described by the peer group's Module Implementation Advertisement are started. In the standard Net Peer Group, this forces the core services to begin providing the Discovery, Resolver, Rendezvous, Peer Info, and Endpoint services. After these service are started, the peer is connected to the network and ready to interact with other peers.

The peer either joins or creates a peer group, after the Net Peer Group was joined. To create a peer group, it uses a Module Implementation Advertisement, which associates with services that this group has and then publishes it.

To join a peer group, the peer finds advertisement of the peer group by using the discovery service. According to the peer group advertisement, the peer can find modules advertisement for the peer group service. After module advertisement is found, the peer loads the peer group services, which are defined in the modules advertisement.

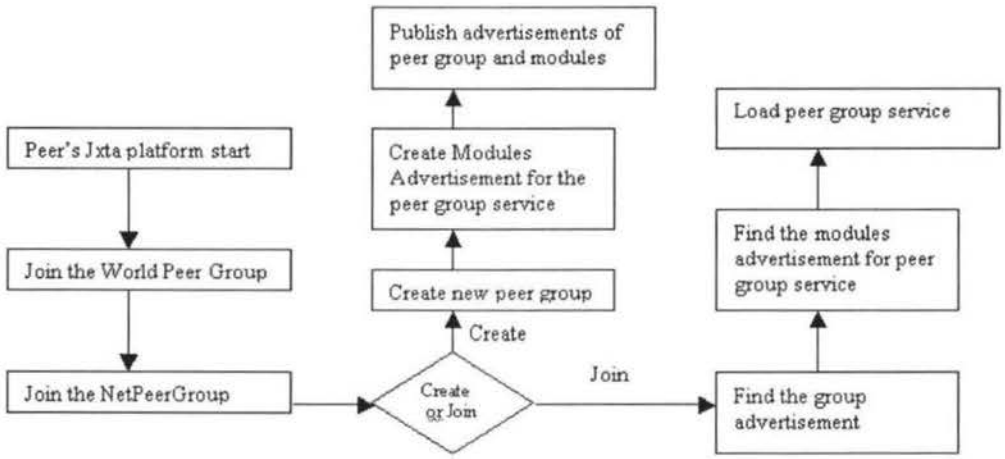


Figure 3.4.2.9 Peer's life Cycle

3.5 Summary

Both the web service and the JXTA projects adopt the Service - Oriented Architecture (SOA), but the communication module is different. The J2EE web service is client-server module, the JXTA project is peer-to-peer module. The OCLE application takes advantage of the SOA and uses the nature of J2EE and JXTA to implement a cooperative learning tool, which centralizes management and distributes conversation.

4 SYSTEM OVERVIEW

The OCLE application was designed for the university's learning environment. University study is often paper based. The students in the class study the same paper and the lecturer teaches this paper. The student can cooperate together to enhance the knowledge of this paper by using this application.

According to the definition of cooperative learning (see introduction section), the core of cooperative learning is that forming a small group and under lecturer's instruction. The following functionalities need to be provided by the OCLE application:

1. The lecturer creates some study groups.
2. The lecturer assigns the student into a group. A student only belongs to one group.
3. Students in a group can discuss and share files on their own schedule.
4. The lecturer can check the performance of each peer group, such as absent status, chat content at later time.

In order to meet those functionalities, the application was divided into two components: the OCLE controlling component and cooperative learning tool component.

4.1 The OCLE controlling component

The OCLE controlling component is a J2EE application. The basic functionalities of this component are:

- Create group.
- Assign the student into group.
- Distribute the group information to cooperative learning tool component.

- Get the group's performance information from cooperative learning tool component.

The OCLE controlling component consists of two parts: the client part and the server part. The client part uses web browser as client side application, which the lecturer can use it to create a new group, assign the student into a group and monitor the performance of each group. The server part is a J2EE application. It uses Jboss as its application server, Tomcat 5.0 as its web server, MySQL as the databases and Jboss.net as the web service framework. It manages and stores all information about student, group and performance, and exposes the group information to cooperative learning tool component.

4.2 Cooperative learning tool component.

The Cooperative learning tool component is a peer-to-peer communication tool, which is a window application. It uses Java as programming language. According to the group information, which has been retrieved from the controlling component, the cooperative learning tool component joins or creates a studying group and students use it to chat and share files to each other within a studying group. The Cooperative learning tool component will send the group performance information to the controlling component every five-minute interval. The advantage of choosing the five-minute interval is not only to avoid increasing network traffic, but also to monitor the peer group's performance.

The cooperative learning tool component uses the JXTA platform, so implementing the complex basic peer-to-peer network protocol was avoided.

Figure 4.2 shows structure of OCLE.

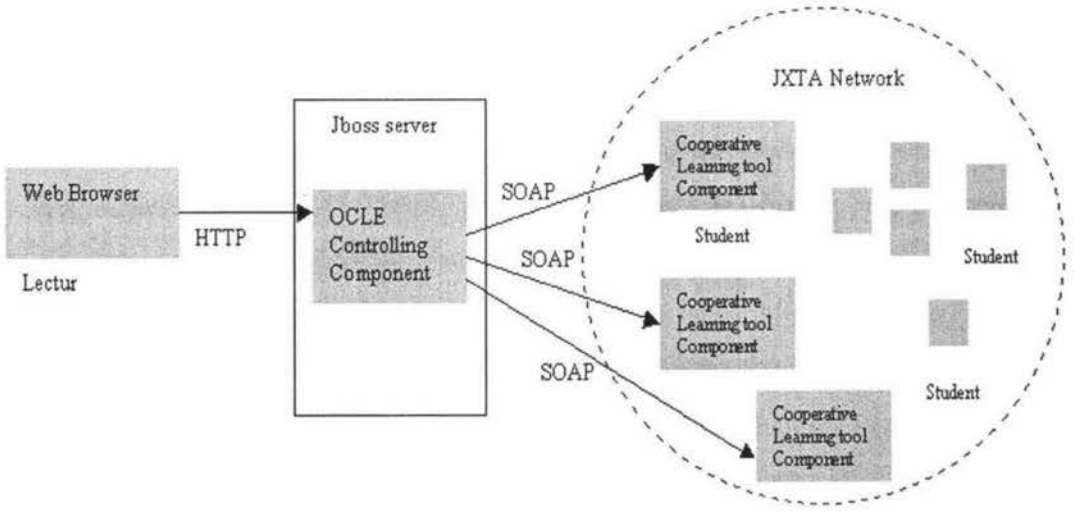


Figure 4.2 Structure of OCLE

5 IMPLEMENTATION

5.1 The OCLE controlling component

According to the component specification of J2EE's (see Section 3.1.1.2), the OCLE controlling component application decomposes into three main components (reusable unit), which are client component, web component and the business logic component.

The client component consists of an applet - GroupEditor object, and two helper classes (StringTransferHandler, ListTransferHandler) for this applet.

The web component consists of two servlets (Dispatcher and LoadGroupByName), a Template Tag Library, 12 jsp pages, and 4 JavaBeans. The web component will be deployed into Jboss web container.

The business logic component consists of two session beans (GroupControllerBean, ReportControllerBean) and three entity beans (ReporterEJB, StudentEJB and GroupEJB). Some methods of two session beans expose as web service. The business logic component will be deployed into Jboss EJB container.

The client component is used by the lecturer to create group, assign student into group and monitor the group's performance. The client component access the student, group and report information maintained in MySQL database through the web component and business logic component. Figure 5.1 shows the architecture of the OCLE controlling component.

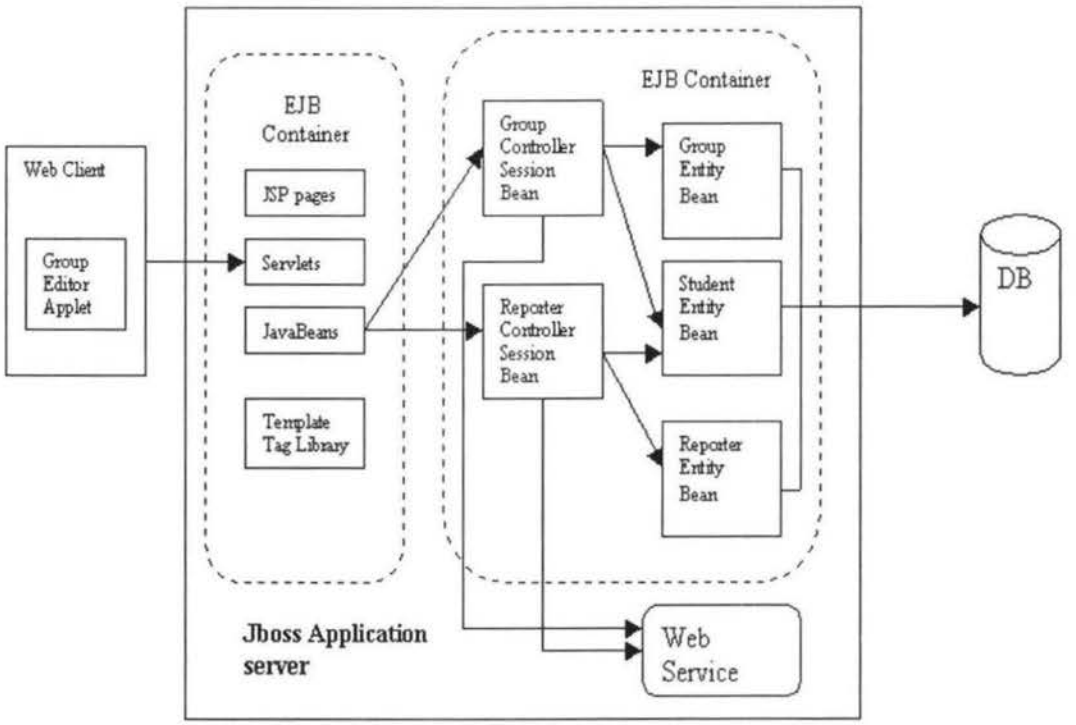


Figure 5.1 Architecture of the OCLE controlling component

The rest of this chapter looks at each of the component types in detail and how these components are deployed into the Jboss application server.

5.1.1 Database

5.1.1.1 Database tables

In order to maintain information about group, student, and group's performance, there are five tables in the MySQL database: student, group, reporter, group_id_xref, and next_reporter_id.

The student table stores the information about a student. The field in the student table includes 'student_name', 'student_id', 'group_name', 'e_mail' and 'absent'. The primary key of this table is the 'student_id', and the foreign key is the 'group_name'.

The reporter table stores the information about a group's performance. The field in the reporter table includes 'timer', 'reporter_num', 'reporter_content' and 'group_name'. The primary key of this table is the 'reporter_num' and the foreign key is the 'group_name'.

The group table stores the information about a group. The field of the group table includes 'group_name', which is the primary key in this table and 'group_id'.
The group_id_xref table stores information about a group ID, which is used by the enterprise bean on later time. The field of the group_id_xref table includes 'group_id' and 'used'. The 'group_id' field stores a group ID, which must follow the rule of the JXTA identifier (see Section 3.4.2.1). There are 30 group Ids, which has been built into this table. The group id was generated by GenerateID object (com.ocle.jxta.help.GenerateID.java). The implementation of the GenerateID class is introduced in the chapter 5.2. Each time, the group ID is used by a new peer group, the 'used' field will be assigned to 'true'. If an existing group is removed, the 'used' field will be assigned to 'false'. Figure 5.1.1.1 shows an example row of the group_id_xref table.

group_id	Used
urn:jxta:uuid-A87A7DD0762F47E88B2FB5452D47B3A802	False

Figure 5.1.1.1 group_id_xref table

The next_reporter_id table holds next primary key for the reporter table. The next_reporter_id table has a single column named id. The value of the id is next primary key that is passed to the create method of an entity bean.

5.1.1.2 Tables relationships

Figure 5.1.1.2 shows the relationships between the database tables. The student and group tables have a many-to-one relationship: A group may have many

students, but each student refers to a single group. The group and reporter tables have a one-to-many relationship: A group may have many reporters, but each report refers to only one group.

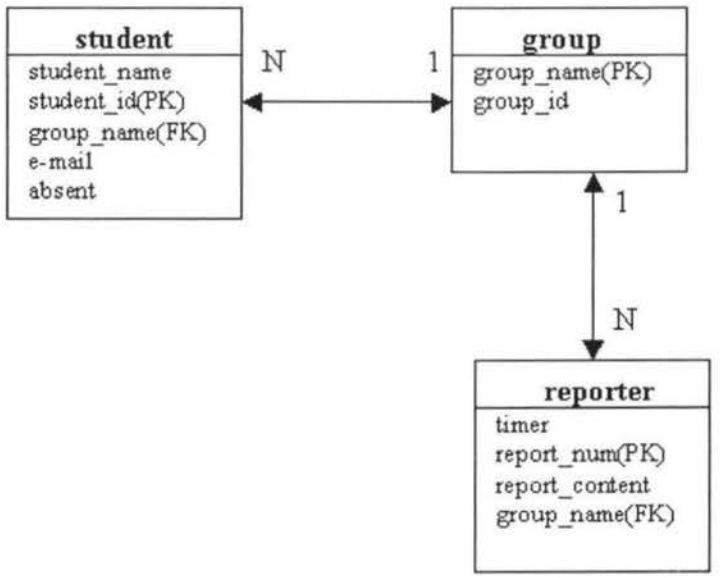


Figure 5.1.1.2 Database Tables in the OCLE application

5.1.1.3 Connection between Jboss and MySQL

The default HSQL database of the Jboss application server 3.2.5 was replaced by the MySQL 4.0 database. The tables, which had been described in section 5.1.1.2 was created in the MySQL database.

Database connection management in the JBoss is entirely handled by the J2EE Connector Architecture (JCA) implementation. So all databases are accessed via JCA resource adapters, which handle connection pooling, security and transactions.

In order to change to MySQL 4.0 and create tables into it, the sequence of operation is as following:

1. Create a database called "jbossdb" within MySQL 4.0 for use by Jboss and create student, reporter, group, next_reperter_id and group_id_xref.

2. Create a user call "jboss" in the user table (it is MySQL 's system table) with password to access the jbossdb database.

3. Create a file called mysql-ds.xml into the Jboss deploy's directory. The mysql-ds.xml is a data source configuration file used by the JCA resource adapter to recognize the MySQL 4.0 database. Figure 5.1.1.3 shows the mysql-ds.xml.

```
<datasources>

  <local-tx-datasource>
   <jndi-name>DefaultDS</jndi-name>
   <connection-url>
      jdbc:mysql://localhost:3306/jbossdb
   </connection-url>
   <driver-class>com.mysql.jdbc.Driver</driver-class>
   <user-name>jboss</user-name>
   <password>password</password>
  </local-tx-datasource>
</datasources>
```

Figure 5.1.1.3 Mysql-dx.xml

5.1.2 Business Logic component

5.1.2.1 Entity Beans

There are three entity beans in the OCLE controlling component: StudentEJB, GroupEJB, ReportEJB. The purpose of these beans is to provide an object's view of these database tables: student, reporter and group (see Figure 5.1.2.1). Because these entity beans use Bean-Managed Persistence, the entity beans contain the

SQL statements that access the tables. Although writing the SQL statements is an additional responsibility, the accessing database gets more control.

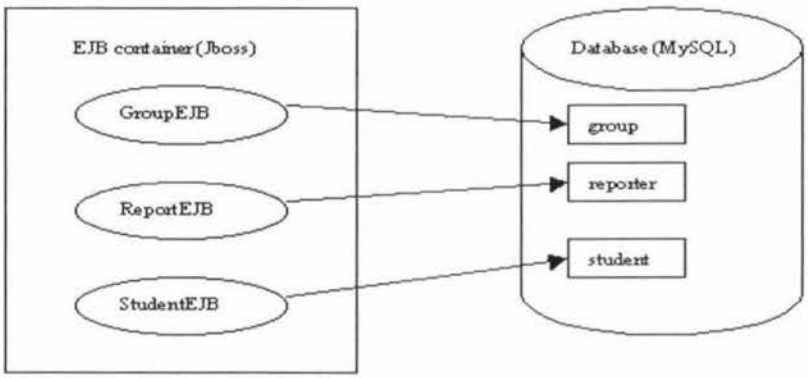


Figure 5.1.2.1 the relationship between entity beans and database tables

The com.ocle package has been chosen in readiness for commercializing this system.

GroupEJB (com.ocle.group.GroupEJB)

The GroupEJB entity bean accesses and manipulates the table 'group'. It provides following business methods, which the session beans can invoke through the remote interface (com.ocle.group.Group) of the GroupEJB.

getPeerGroupID

The getPeerGroupID method returns the peer group id.

getGroupDetails

The getGroupDetails method returns a GroupData object containing the group instance variables.

The GroupEJB provides following methods, which the session bean can create and find through the home interface (com.ocle.group.GroupHome) of the GroupEJB.

ejbCreate

The ejbCreate method is used to create a group entity bean instance. It inserts a new row, which includes group name and group ID into the 'group' table.

remove

The remove method is used to remove an existing row from the 'group' table.

ejbfindByPrimaryKey

The ejbfindByPrimaryKey method returns the primary key of the GroupEJB entity bean according to the input parameter – Group's name.

ejbFindByAll

The ejbFindByAll method returns all groups, which is in the 'group' table.

ReportEJB (com.ocle.reporter.ReportEJB)

The ReportEJB entity bean accesses and manipulates the table 'reporter'. It provides following business methods, which the session beans can invoke through the remote interface (com.ocle.reporter.Report) of the ReportEJB.

getReportContent

The getReportContent method returns the reporter's content.

getReportTimer

The getReportTimer method returns the time of the reporter.

getReportGroup

The getReportGroup method returns the group name for a reporter. The ReportEJB provide following methods, which the session bean can create and find through the home interface (com.ocle.reporter.Home) of the ReportEJB.

ejbCreate

The ejbCreate method is used to create a ReportEJB entity bean instance. It takes 4 parameters as the input. The 4 input parameters are reporter's number, timer, group's name and reporter's content. The ejbCreate method inserts a new row into the 'reporter' table with these input parameters.

ejbFindByGroupName

The ejbFindByName method returns a collection of primary key, which matches a given group's name.

StudentEJB (com.ocle.student.StudentEJB)

The StudentEJB entity bean accesses and manipulates the table 'student'. It provides following business methods, which the session beans can invoke through the remote interface (com.ocle.student.Student) of the StudentEJB.

emptyPeerGroupField

The emptyPeerGroupField method sets the group_name field of the student table into null by using private *updataGroupName* method, which issues the SQL updata statement.

AssignIntoGroup

The AssignIntoGroup method sets the group_name field of the student table into a given name by using private updataGroupName method.

getGroupName

The getGroupName method returns the group's name.

getAbsent

The getAbsent method returns the number of absence. The StudentEJB provides the following methods, which the session bean can create and find through the home interface (com.ocle.student.StudentHome) of the StudentEJB.

ejbFindByPrimaryKey

The ejbFindByPrimaryKey method returns a primary key, which matches a given student ID.

ejbFindByPeerGroup

The ejbFindByPeerGroup method returns a collection of primary key, which matches the given group name.

ejbFindByUnssign

The ejbFindByUnssign method returns a collection of student, which are not assigned into any peer group which means the group_name field of student table is empty.

ejbFindByName

the ejbFindByName method returns a collection of student, which matches a given student's name.

Figure 5.1.2-2 shows the ReportEJB entity bean in UML.

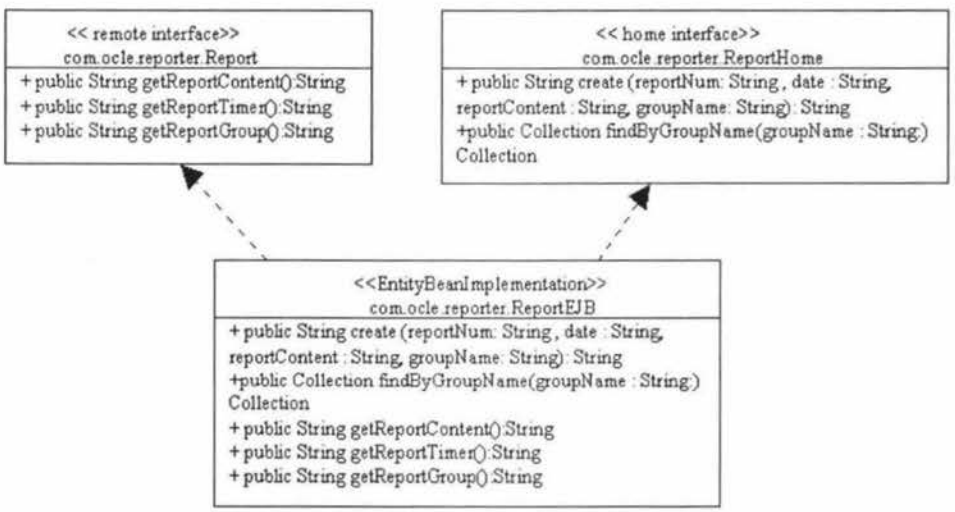


Figure 5.1.2-2 ReportEJB entity bean in UML

5.1.2.2 Session Beans

The OCLE controlling component has two session beans: ReportControllerEJB and GroupControllerEJB. These session beans provide a client's view of the OCLE controlling component's business logic. The server-side routines, which implement the business logic, access databases, manage relationships, and perform error checking, are hidden from client component.

5.1.2.2.1 GroupControllerEJB

The GroupControllerEJB session bean handles the following tasks:

1. Create the new group or remove the existing group.
2. Assign the student into a studying group or remove the student from a studying group.
3. Provide the information about the group, the student and the member of a study group.

In order to perform these tasks, the GroupControlleEJB session bean provides the following business methods, which the web component can invoke through remote interface (com.ocle.group.GroupController) of the GroupcontrollerEJB session bean.

createPeerGroup

The createPeerGroup method is used to create a new peer group with group name and group ID. It calls *getGroupID* method of the *DBHelper* object to get the peer group ID and then calls the *create* method of the GroupEJB entity bean to store the new group information. The createPeerGroup method also verifies that the specified group name exists by invoking the *findByprimaryKey* method of the GroupEJB entity bean. If the result of this verification is true, the createPeerGroup throws a *DuplicateGroupNameException*.

removePeerGroup

The romovePeerGroup method is used to delete the existing group. It calls *remove* method of the GroupEJB entity bean to remove a given group from database. Then this method calls the *releasePeerID* method of the DBHelper class to make this group ID available again. Finally, the removePeerGroup method calls

findbyPeerGroupName method of the StudentEJB entity bean to get all students of this deleted group, and then calls *emptyPeerGroupField* method of the StudentEJB entity bean to remove the student from this group.

addIntoGroup

The addIntoGroup method is used to assign a student into a given group. It calls *findByPrimaryKey* method of the StudentEJB entity bean to find a student from 'student' table and then calls the *assignIntoGroup* method of the StudentEJB entity bean to set the group name to this student.

removeStudentFromGroup

The removeStudentFromGroup method is used to remove a student from a given group. It calls the *findByPrimaryKey* method of the StudentEJB entity bean to find a student from 'student 'table and then to call the *emptyPeerGroupField* method of the StudentEJB entity bean to remove this student from the given group.

getOneGroupStudent

The getOneGroupStudent method returns all of the students of the given group by invoking the *findByPeerGroup* method of the StudentEJB entity bean.

getGroupInfoByStudent

The getGroupInfoByStudent method returns the GroupData (see Section 5.1.2.3) object of the given student. It calls the *findByPrimaryKey* method of the StudentEJB entity bean to find a student and then calls the *getGroupName* method of the StudentEJB entity bean to get a group' name for this student. Finally, it calls the *getGroupDetails* method of the GroupEJB entity bean.

The *getGroupInfoByStudent* method is exposed as web service. The GroupData object is treated as a Java bean, which can be handled by the Jboss.net bean serialization classes.

getUnassignStudent

The getUnassignStudent method returns all of the students, which are not assigned to any studying group by invoking *findByUnAssign* method of the StudentEJB entity bean.

getGroupList

The getGroupList method returns all the existing groups by invoking the *findByAll* method of the GroupEJB entity bean .

getStudentAbsent

The getStudentAbsent method returns total number of absence for a given student by invoking *getAbsense* method of the StudentEJB entity bean.

The home interface of GroupControllerEJB session bean is GroupControllerHome (com.ocle.group.GroupControllerHome) and the remote interface of GroupEJB entity bean is GroupController (com.ocle.group.GroupController).

Figure 5.1.2.2-1 shows a structure Diagram of the GroupControllerEJB session bean. The client in this diagram is the web component. The client creates the GroupControllerEJB session bean through the GroupControllerHome and invokes the business method through the GroupController (remote interface for the GroupControllerEJB session bean).

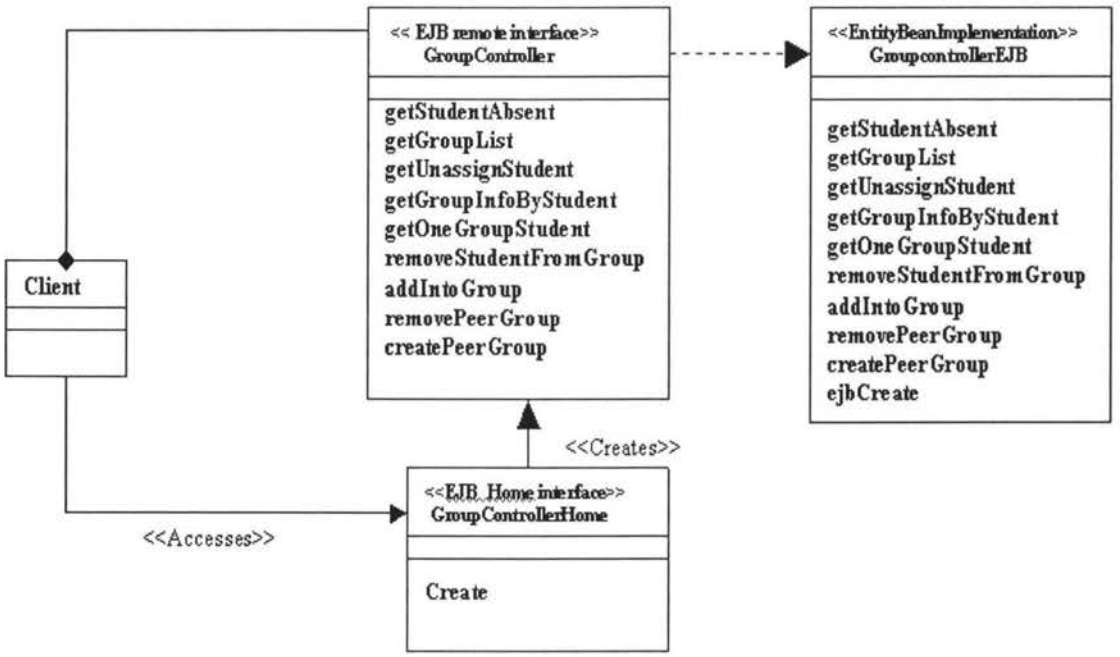


Figure 5.1.2.2-1 The architecture of the GroupControllerEJB session bean

5.1.2.2.2 ReportControllerEJB

The ReportControllerEJB responses for

1. According to the information, which is received from the OCLE cooperative tool, the ReportControllerEJB component creates a reporter of a group performance.

2. Providing the reporter of a group performance.

The home interface of the ReportControllerEJB session bean is ReportControllerHome (com.ocle.report.ReportControllerHome) and the remote interface of the ReportControllerEJB entity bean is ReportController (com.ocle.report.ReportController).

In order to implement these functionalities, the ReportControllerEJB session bean provides the following business methods:

createReport

The createReport method is exposed as the web service. It is invoked by the OCLE cooperative learning tool. It parses the OnlineData (see Section 5.1.2.3) object, which contains the group's name, list of the currently online members, and sending time into local variable. Then it calls *ejbFindByPeerGroup* method of the StudentEJB entity bean to get a list of the group's member. It compares those two lists to generate a list of the absent student by invoking the private *compareStudentList* method. It calls *getReportID* method of the DBHelp class to get a reporter' number. It calls the *create* method of the ReportEJB entity bean with the time, the group's name, the absent list of the student and the reporter's number to stores a reporter into the 'reporter' table.

According to the list of the absent students, the createReport method calls the *addAbsent* method of the StudentEJB entity bean to increment the absent value, which matches the given student.

getReportDetails

The ReportDetails method returns a list of the ReportData object (see Section 5.1.2.3) for a given group. It calls the *findByGroupName* method of the GroupEJB entity bean to get a list of the reporter for a given group. It calls *getReportContent* and *getReportTimer* methods of the GroupEJB entiy bean to get the time and the reporter's content. It encapsulates the time and the reporter's content into the ReportData object.

The home interface of GroupControllerEJB session bean is GroupControllerHome (com.ocle.group.GroupControllerHome) and the remote interface of GroupEJB entity bean is GroupController (com.ocle.group.GroupController).

Figure 5.1.2.2-2 shows a structure Diagram of the ReportControllerEJB session bean.

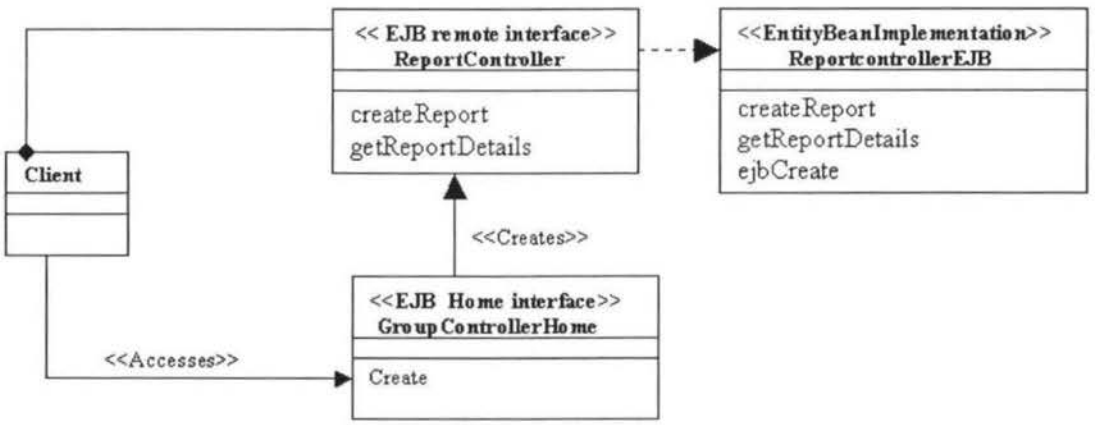


Figure 5.1.2.2-2 The architecture of the ReportControllerEJB session bean

5.1.2.3 Helper Classes

GroupData

The GroupData class encapsulates the state of a GroupEJB instance. It is returned by the *getGroupDetails* method of the GroupEJB entity bean and the *getGroupInfoByStudent* method of the GroupControllerEJB session bean. The GroupData object is declared in the web service descriptor file under the 'beanMapping' element.

OnLineData

The OnLineData class encapsulates the information of the group performance, which includes the time for a reporter, group' s name and reporter's content. The OnLineData object constructs in the OLCE cooperative learning tool and parses in the createReport method of the ReportController session bean. It is declared in the web service descriptor file (web_service.xml) under 'beanMapping' element.

ReportData

The ReportData class encapsulates the state of a ReportEJB instance. It is returned by the *getReportDetails* method of the ReportControllerEJB session bean.

DBHelper

The DBHelper class has three methods:

getReportID

The getReportID method reads the id from the next_reporter_id table, increments the id value in the table and returns the id.

getGroupID

The getGroupID method reads the group id from the group_id_xref table, if the used value is false. It sets the used value into true in the table and returns the group id.

releasePeerID

The releasePeerID method reads the used value from the group_id_xref table, which matches a given group id and sets used value into false in the table.

5.1.2.4 Packaging into Module

The GroupEJB entity bean, StudentEJB entity bean, ReporterEJB entity bean, GroupControllerEJB session bean and ReportController session bean are packed into three modules [18].

The group module (group-ejb.jar) includes all home interface, remote interface and bean class of the GroupEJB entity bean and GroupControllerEJB session bean. It includes an EJB descriptor (group-ejb.xml), which provides both the structural and application assembly information for those beans in the EJB module.

The student module (student-ejb.jar) includes all home interface, remote interface and bean class of the Student entity bean. It includes the student-ejb.xml as the EJB descriptor file.

The Reporter module (reporter-ejb.jar) includes all home interface, remote interface and bean class of the ReportEJB entity bean and the ReportControllerEJB session bean. It includes the reporter-ejb.xml as the EJB descriptor file.

5.1.2.5 Web Service

In order to expose the GroupControllerEJB and ReportControllerEJB session beans as web service, a descriptor of the web service module need to be created and then the web service module is built, which includes the descriptor file (web-service.xml). The web service module file (groupcontroller.wsr) is just a standard Jar archive with a .wsr extension.

Figure 5.1.2.5 shows the descriptor file of the web service module.

```
<deployment
  name="ocle"
  xmlns="http://xml.apache.org/axis/wsdd/"
  targetNamespace="http://net.jboss.org/ocle"
  xmlns:bank="http://net.jboss.org/ocle"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="GroupController" provider="Handler">
  <parameter name="handlerClass" value="org.jboss.net.axis.server.EJBProvider"/>
  <parameter name="beanJndiName" value="MyGroupController"/>
  <parameter name="allowedMethods" value="getGroupInfoByStudent"/>
 </service>
 <service name="ReportController" provider="Handler">
  <parameter name="handlerClass" value="org.jboss.net.axis.server.EJBProvider"/>
  <parameter name="beanJndiName" value="MyReportController"/>
  <parameter name="allowedMethods" value=" createReport "/>
 </service>

 <beanMapping qname="ocle:GroupData"

        languageSpecificType="java:com.ocle.ejb.util.GroupData"/>

 <beanMapping qname="ocle:OnlineData"
        languageSpecificType="java:com.ocle.ejb.util.OnlineData"/>
</deployment>
```

Figure 5.1.2.5 web-service.xml

The illustration of the web-service.xml:

There are two services available, which are the GroupController and the ReportController. (Defined by name attribute of service tag). For the GroupController service, the Jndi's name is MyGroupController, which maps to the GroupControllerEJB session bean; the org.jboss.net.axis.server.EJBProvider class is responsible for handling the details; the *getGroupInfoByStudent* method of the GroupControllerEJB session bean is allowed to expose. For the ReportController service, the Jndi's name is MyReportController, which maps to the ReportControllerEJB session bean; the org.jboss.net.axis.server.EJBProvider class is responsible for handling the details; the *createReport* method of the ReportControllerEJB session bean is allowed to expose.

The final beanMapping element specifies that the GroupData and OnlineData objects. They should be treated as a Java bean, which the (de)serialization to and from SOAP messages will be handled by the Jboss.net bean serialization classes.

Figure 5.1.2.5-1 shows the structure of web service.

The GroupControolerEJB and ReportControllerEJB session beans represent the service interface to the GroupEJB, StudentEJB, ReportEJB entity beans and are mapped into a SOAP service via the Jboss.net framework

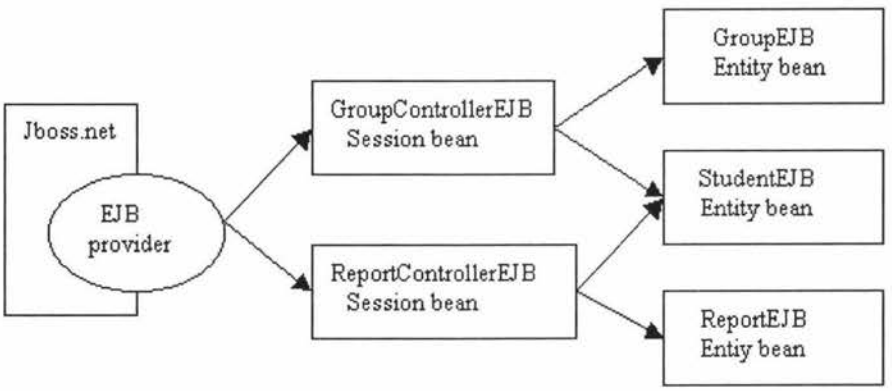


Figure 5.1.2.5-1 Architecture of the web service

Once the web service component (groupcontroller.wsr) is deployed, the WSDL file for the GroupController service is available on the URL http://192.168.1.3:8080/jboss-net/services/GroupController?wsdl, and the WSDL file for the ReportController service is available on the URL http://192.168.1.3:8080/jboss-net/services/ReportController?wsdl.
The IP address can be configured for the host where the OCLE is deployed.

The two URL address will be built into the OCLE cooperative learning tool. The OCLE cooperative tool can use those URL to load the WSDL files and invoke those web services.

5.1.3 Web component

The web component is used by the lecturer to create the studying group, assign the student into a studying group and check each group's performance. In order to

implement those functionalities, the web component uses the Front controller pattern to design the web component.

The Front controller pattern centralizing controls the client's request and forwards the request to the next view [20]. Using the Front controller design pattern, the client can get common look across the JSP pages.

According to the Front controller design pattern, the web component discomposes into four components, which is the Controller, the Dispatcher, the View and the Helper.

Figure 5.1.3 shows the architecture of the web component in the Front controller design pattern.

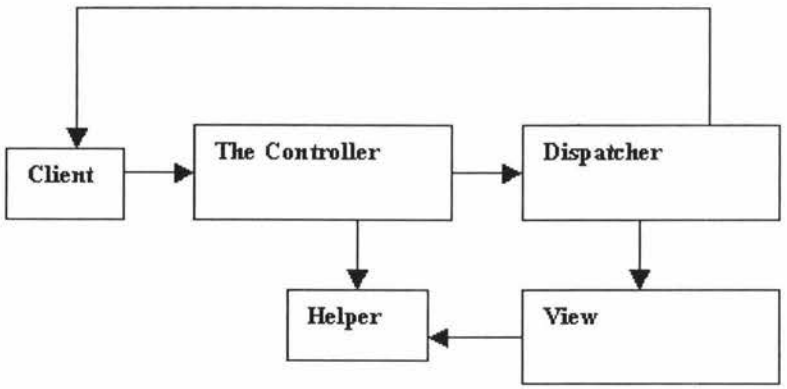


Figure 5.1.3 The Front Controller Design pattern

A controller handles all requests, it is an initial point for handing a request. The controller manages content retrieval by invoking the Helper and delegates to a Dispatcher component. The helper component is a façade to the interface provided by a session bean and helps the controller or the view to complete its processing. The dispatcher component is responsible for viewing management and navigation. The view component represents and displays information to client.

5.1.3.1 Helper component

In the web component of this application, the helper consists of the JavaBeans, which includes BeanManager, AppletList, AssignIntoGroup, ViewAck.

5.1.3.1.1 BeanManager (com.ocle.web.BeanManager.java)

The BeanManager object responses for managing the enterprise beans, which are used by the web client. It creates the GroupControllerEJB session bean and the ReportControllerEJB session bean and provides *getGroupController* and *getReportController* methods for retrieving the beans.

After instantiated, the BeanManager object retrieves the home interfaces for each bean from EJBGet object (see Section 5.1.3.5). It creates an instance by calling the *create* method of the home interface.

When the client is first initialized, the BeanManager is created and stored as a context attribute by a ContextListener.

5.1.3.1.2 AppletList (com.ocle.web.AppletList.java)

The AppletList class is used to retrieve a list of the all students, a list of all groups from the GroupControllerEJB session bean. The GroupEditor object (Applet) will use all these informations. The AppletList class provides the following methods.

Contractor

After instantiated, it retrieves the GroupControllerEJB's remote interface by using the *getGroupController* method of the BeanManager object.

processing

The processing method uses the *getGroupList* method of the GroupControllerEJB session bean to get a list of group and uses *getUnassignStudent* method of the GroupControllerEJB to get a list of the unassigning student. It uses *getOneGroupStudent* method of the GroupControllerEJB session bean to get a list of student within a group.

getlistOfAllstudent

returns a list of student.

getListOfstudent

returns a list of student within a group.

getListOfGroup

returns a list of group.

5.1.3.1.3 AssignIntoGroup (com.ocle.web.AssigIntoGroup)

The AssignIntoGroup class handles a group member's change. It compares the current group member with the original one and stores the group member's change into the database through the GroupControllerEJB session bean. The GroupEditor (Applet) object will use it to communicate with session bean.

setBeanManager

set the BeanManager bean object.

setStudent

set a list of the student within a group.

setGroup

set a group name.

processing

The processing method compares the current group member with the original one and saves the change. It calls the *getOneGroupStudent* method of the GroupControllerEJB session bean, which matches a given group name to get the original list of group member. It compares the original list with the current list of student. if any change happen, the processing method uses the *addIntoGroup* method of the GroupControllerEJB session bean to add a student into a group or uses the *removeStudentFromGroup* method of the GroupControllerEJB session bean to remove a student from a study group.

5.1.3.1.4 ViewAck (com.ocle.web.ViewAck)

The ViewAck object is used to retrieve a list of reporterData object, which includes a reporter's time, reporter's content for a given group.

setBeanManager

set a BeanManager object.

setGroup

set a group name.

getReportDetails

returns a list of ReportData object.

processing

The processing method calls the *getReportDetails* method of the ReportControllerEJB session bean to get a list of the ReportData object.

Figure 5.1.3.1 shows the communication between the session beans and the help components.

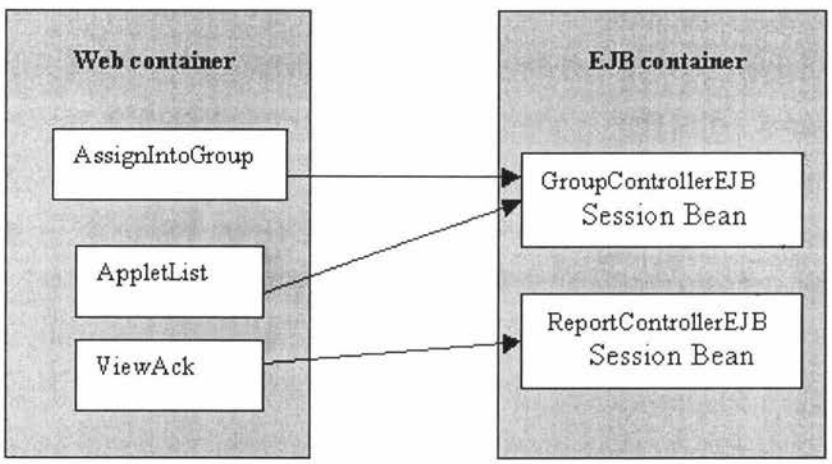


Figure 5.1.3.1 the structure of the Helper

5.1.3.2 The Controller

Dispatcher (com.ocle.web.Dispacher.java)

The Controller component is implemented by the Dispatcher servlet object (com.ocle.web.Dispatcher.java). The Dispatcher object acts as an initial point for the client's request. All client's requests are going into the Dispatcher object. When a request is delivered to Dispatcher object, if the request's URL is "/editGruop" or "/viewReportAck" or "/assignIntoGroup" or "/viewReport":

1. Retrieves and saves the incoming request URL in the request attribute selectedScreen.
2. Creates a JavaBeans component and stores the bean as a request attribute according to the selectedScreen attribute.
3. Parses and validates the request parameters. If a parameter is invalid, the Dispatcher object resets the request alias to an error page.
4. Calls the processing method of the JavaBeans component. This method retrieves data from the enterprise beans and processes the data.
5. Forwards the request to the dispatcher components. If the URL is "/assignIntoGroup", the forwards action will not happen.
   The table 5.1.3.2 shows the relationship between request's URL and the JavaBeans.

URL Aliases	**JavaBeans**
/editGruop	**AppletList**
/viewReportAck	**ViewAck**
/assignIntoGroup	**AssignIntoGroup**
/viewReport	**ViewAck**

Table 5.1.3.2 URL Aliases and JavaBeans

If the request's URL is "/groupAdded", the Dispatcher object does not invoke any JavaBeans. It calls the *createPeerGroup* method of the GroupControllerEJB session bean by using the BeanManager object to create a new group. It sets a new attribute 'added' into request. It resets the request's URL to '/addGroup' and forwards to the dispatcher component.

If the request's URL is "/loadGroup", the Dispatcher object calls the getOneGroupStudent mehod of the GroupControllerEJB session bean to get list of the student, which is in same group. It sends response back to client, which is the GroupEditor applet.

If the request's URL is "/assignIntoGroup", another request's URL, which do not include in the table 5.1.3.2, are forwarded to dispatcher component directly. Figure 5.1.3.2 shows the operation of the front controller component.

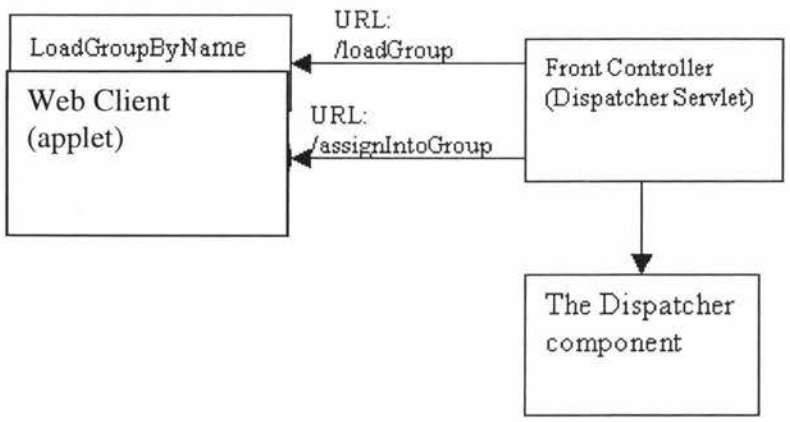


Figure 5.1.3.2-2 The operation of the front controller component.

5.1.3.3 Dispatcher component

According to the request's URL, which is derived by the controller component, the dispatcher component loads the JSP pages, combines into a screen, returns the screen to the client. The dispatcher component provides a template mechanism to maintain and enforce a consistent look and feel in all the screens.
The dispatcher component of the web component consists of the viewTemplate.jsp, screenDefine.jsp and a template tag library.

5.1.3.3.1 Template Tag Library

The template tag library provides the four nested tags, which are screen, definition, parameter and insert. The screen tag is handled by the ScreenTag (com.ocle.web.taglib.ScreenTag.java) tag handler; the definition tag is handled by the DefinationTag (com.ocle.web.taglib.DefinationTag.java) tag handler; the parameter tag is handled by the ParameterTag

(com.ocle.web.taglib.ParameterTag.java) tag handler and the insert tag is handled by InsertTag (com.ocle.web.taglib.InsertTag.java) tag handler.

These nested tags are used by the viewTemplate.jsp and screenDefine.jsp to create a common look screen for client.

viewTemplate.jsp

The viewTemplate.jsp determines the structure of each screen. It includes the screenDefine.jsp, which creates the screen *definition* and uses *insert* tag to construct a screen from view component.

screenDefine.jsp

The screenDefine.jsp defines the view component used by each screen. It creates a screen *defination* based on a request attribute *selectedScreen*. All screen have the same banner, but they have different title and body content.

5.1.3.4 View component

The view component consists of following JSP pages.

login.jsp

The user can use the login.jsp page to log on to application.

addGroup.jsp

The user can use this page to create a new group and view a list of the existing groups. It calls the *getGroupList* method of the GroupControllerEJB session bean to get the list of the group name by invoking the BeanManage object. It uses the

submit form to submit the new group to URL '/groupAdded'. It checks the request's attribute 'added', if the 'added' is not empty, it displays a group added message.

error.jsp

The error.jsp dispays a error message, when operation's error happens. It retrieves the error message from the request's attribute-errorMessage and displays it.

linkAnother.jsp

The linkAnother.jsp provides the links to another screen.

banner.jsp

The banner.jsp displays page's banner. All screens have the same banner.

logoff.jsp

The user uses the logoff.jsp page to finish the current session and starts a new session.

editGroup.jsp

The user uses the editGroup.jsp page to edit the group member of a group. The GroupEditor applet is embedded into this page. It gets a list of the all students, a list of the existing group and list of student with a group from the AppletList JavaBean, which is set as a request's attribute in the Dispatcher servlet. It sets those lists as the parameters for the GroupEditor applet. Figure 5.1.3.4 shows the applet's setting in the editGroup.jsp.

```
<jsp:plugin
   type="applet"
   code="com.ocle.applet.GroupEditor"
   archive="appletEdit.jar"
   jreversion="1.3"
   align="center" height="60%" width="70%">
       <jsp:params>
           <jsp:param name="listOfAllStudent" value="<%=allStudent%>"/>
              <jsp:param name="listOfStudent" value="<%=memberList%>"/>
              <jsp:param name="listOfGroup" value="<%=groupList%>"/>
              <jsp:param name="base_URL" value="<%=serverUrl%>" />
       </jsp:params>
       <jsp:fallback><p align="left">Unable to start Java Plug-
in.</p></jsp:fallback>
  </jsp:plugin>
```

Figure 5.1.3.4 editGroup.jsp

viewReport.jsp

The viewReport.jsp page displays a list of group. If any of these groups is clicked, the request will send to URL '/viewReportAck' with the group's name.

5.1.3.5 Operation of the Front control's component.

The client sends the different request's URL. The web component performs the different function to support it and the client will get different screen. The screen consists of the JSP pages. The presentation logic is performed by the Dispatcher component and the Control component. The table 5.1.3.5 lists the functions, the request' URL which is used to access the functions and the component that implements the functions.

Function	URL Aliases	JSP pages and Servlet	JavaBeans Component
Log on the application	**/logon**	**Banner.jsp login.jsp**	--
Log off the application	**/logoff**	**Banner.jsp linkAnother.jsp. logoff.jsp**	
Assign student into	**/assignIntoGroup**		**AssignIntoGroup**

a group			
Get a group member Within a group	/loadGroup		
Create a new group	/addGroup /groupAdded	banner.jsp. addGroup.jsp linkAnother.jsp	
Load a GroupEditor applet	/editGroup	banner.jsp linkAnother.jsp editGroup.jsp	AppletList
View a report	/viewReport	banner.jsp linkAnother.jsp viewReport.jsp viewReportAck.jsp	ViewAck

Table 5.1.3.5 Web component and functionalites

Figure 5.1.3.5 shows a client's screen for '/addGroup'

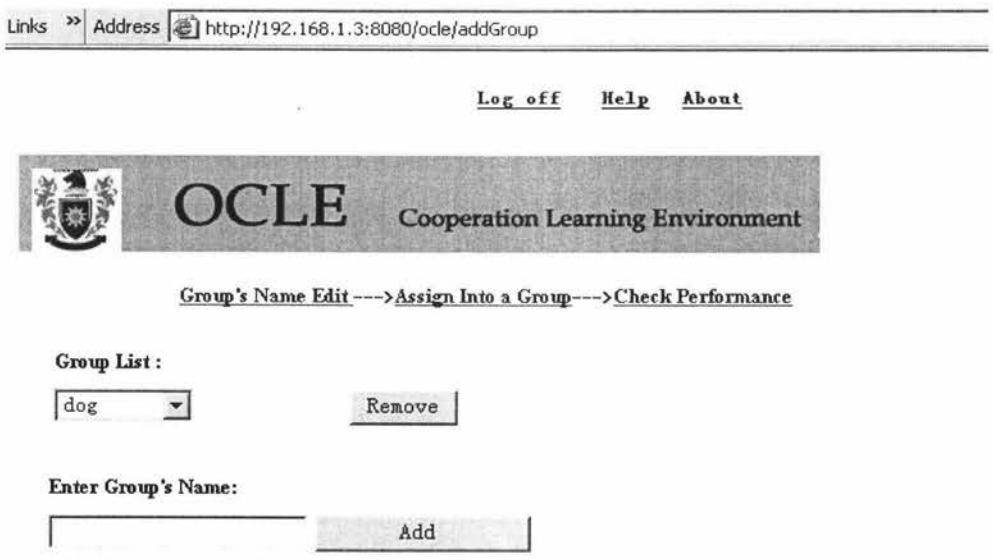


Figure 5.1.3.5 create a new group

Figure 5.1.3.5 shows a client's screen of '/viewReport'.

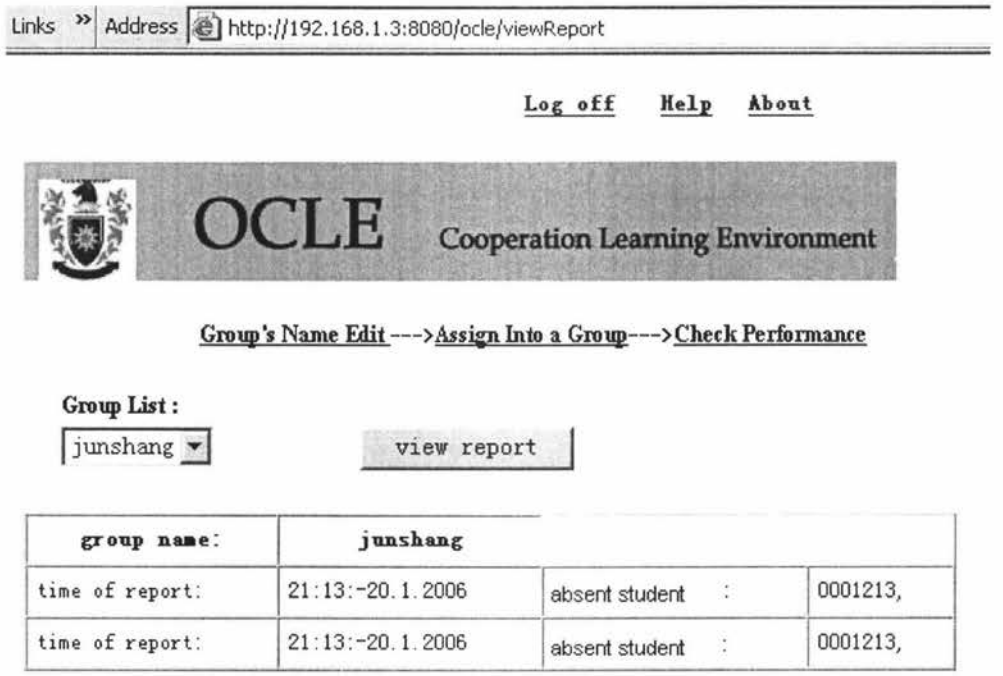


Figure 5.1.3.5-1 View a report

To log on to this application is deferent from others. It uses the security manager component of the Jboss application server to handle the security issue.
A security domain is created for the web component of this application, namely "OCLEDomain". What this means is that JBoss will bind a security manager instance for this application under the JNDI name "java:/jaas/OCLEDomain".

The OCLEDomain is added into the descriptor of the web component (web.xml). The security manager of the Jboss application server protects all web component of this application.

In order to tell the security manager of the OCLEDomain about the security rule, the user.properties and roles.properties will be built into the web component. The user log on to the application by sending '/login' request to the Dispatcher servlet. The Dispatcher servlet forwards the request to the security manager of the OCLEDomain. The security manager will check the user.properties and

roles.properties files to valid this request. If the validation is unsuccessful, the '/loginError' request will be sent. Figure 5.1.3.5-2 shows the login processing.

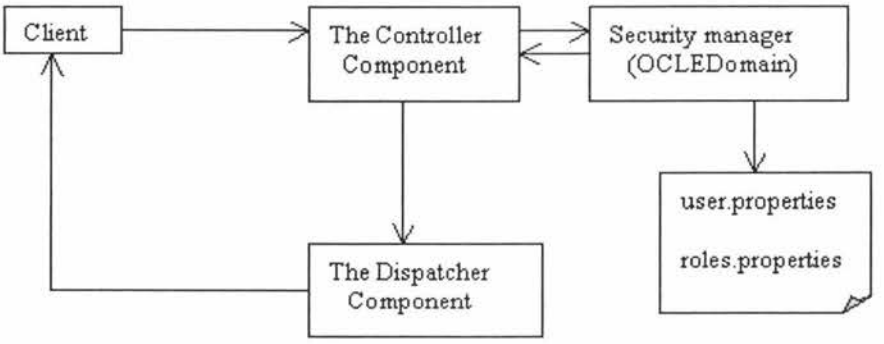


Figure 5.1.3.5-2 The user log on to the OCLE controlling component

5.1.4 Applet – GroupEditor (com.ocle.web.applet)

The GroupEditor applet object provides a sophisticated enough user interface (UI). The lecture uses it to assign the student into a group or remove the student from a group. It is embedded into the editGroup.jsp (see Section 5.1.3.4).

Figure 5.1.4 shows the view of the GroupEditor applet. See Figure 5.1.4.1-2 for a screen shot.

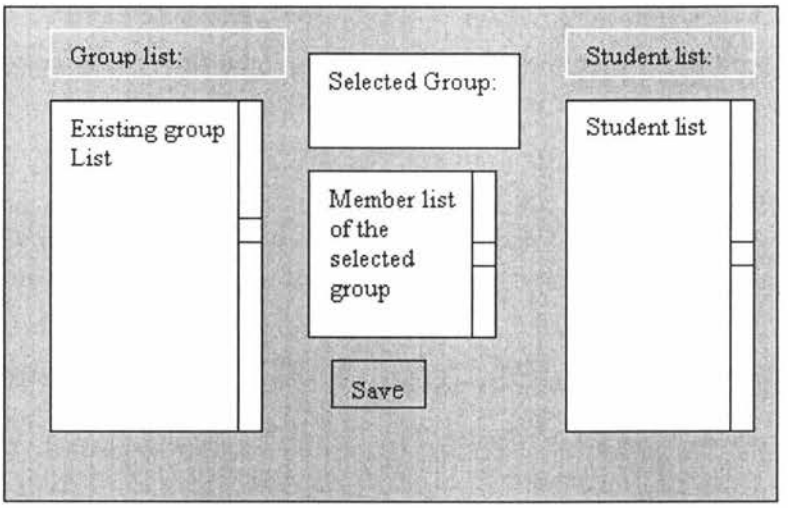


Figure 5.1.4 GroupEditor applet

The user selects a group from the existing group list. The group name will be displayed in the Selected Group label and the member list will display in the Member list. The user adds a student into this selected group by selecting a student from student list, drag and drops it into the member list. The user clicks the save button to save the change.

5.1.4.1 GroupEditor implementation

The GroupEditor consists of the three JLists, a Jbutton and three Jlables. It uses java.awt.GridBagLayout as a layout manager. Because the JList does not support the drag and drop operation, the custom TransferHandlers (com.ocle.web.applet. ListTransferHandler, com.ocle.web.applet.StringTransferHandler) will be implemented to deal with the actual transfer of data.

After instantiated, the GroupEditor applet gets a list of the groups, a list of the students, a list of the members within a group from the applet parameters, which was assigned in the editGroup.jsp page (see Section 5.1.3.4). The three JList object are constructed by those list.

An ActionListner object is created for the save button and a ListSelectionListener object is created for the JList of the group list.

The following two methods implement the functionality that the GroupEditor applet communicates with the service side components.

memberListStore

when the user clicks the save button or selects a new item in group list , the memberListStore method is triggered.

The memberList method gets all items of the member list and are convert into a string, which each item was separate by ','. It creates a URL query string, which contains the member list string and the group name. The URL query string is: *HOST+"/ocle/assignIntoGroup?student="+send "&selectGroup="+currentGroup;*. It sends it to the web component.

Figure 5.1.4.1 shows a sequence diagram for the memberListStore.

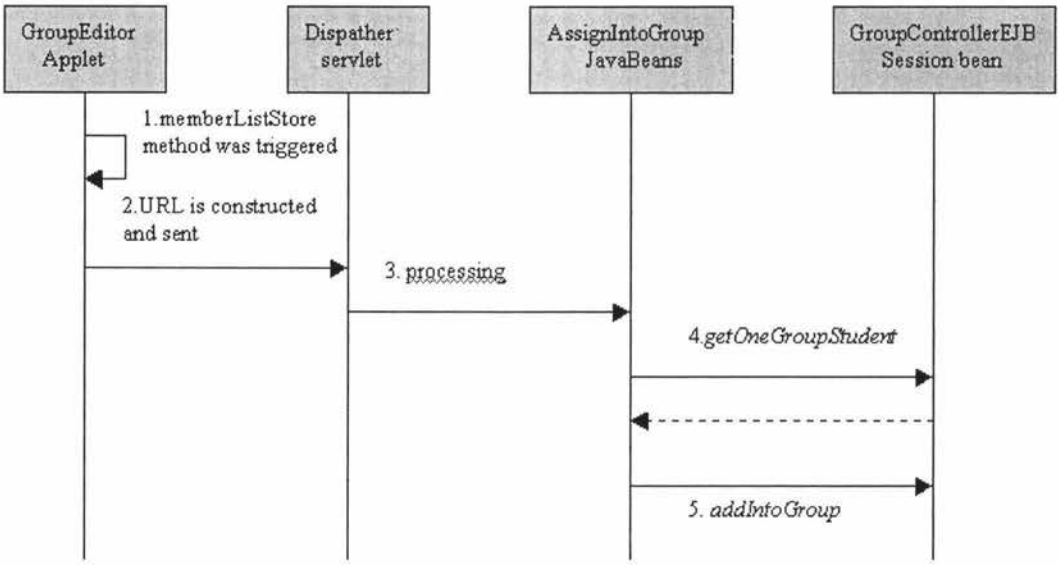


Figure 5.1.4.1 a sequence diagram for the memberListStore

memberListReload

The memberListReload is triggered, when the user selects a new item in the group list.

The memberListReload method calls the *getElementAt* method to get the group name. It creates a URL query string, which contains the group name. The URL query string isHOST+"/ocle/loadGroup?selectGroup="+currentGroup. It sends it to the web component.Figure 5.1.4.1-1 shows a sequence diagram for the memberListReload.

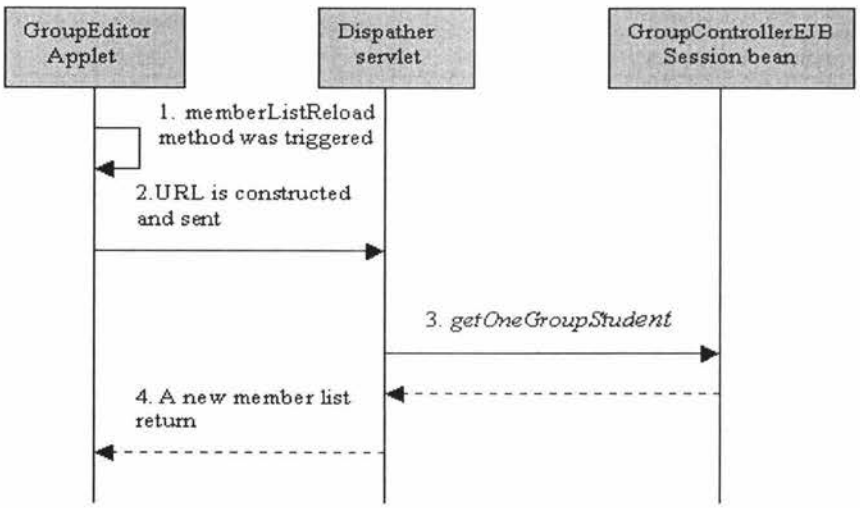


Figure 5.1.4.1-1 A sequence diagram for the memberListReload

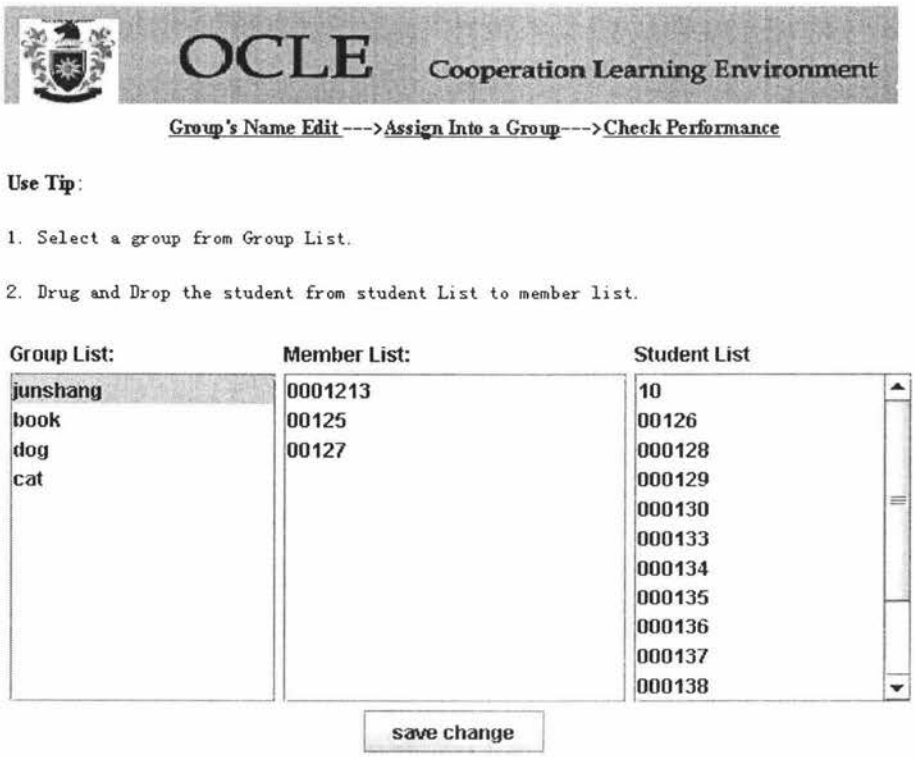


Figure 5.1.4.1-2 the GroupEditor applet

5.2 Cooperative Learning Tool (CLT)

5.2.0 Cooperative learning tool Architecture

The design of Cooperative learning tool (CLT) complements the JXTA software architecture by building upon the core services provided (see Section 3.3.1) and introducing two new services that reside in the service layer. These new services are utilized in turn by the CLT application that resides in the application layer. Figure 5.2 illustrates the CLT software architecture.

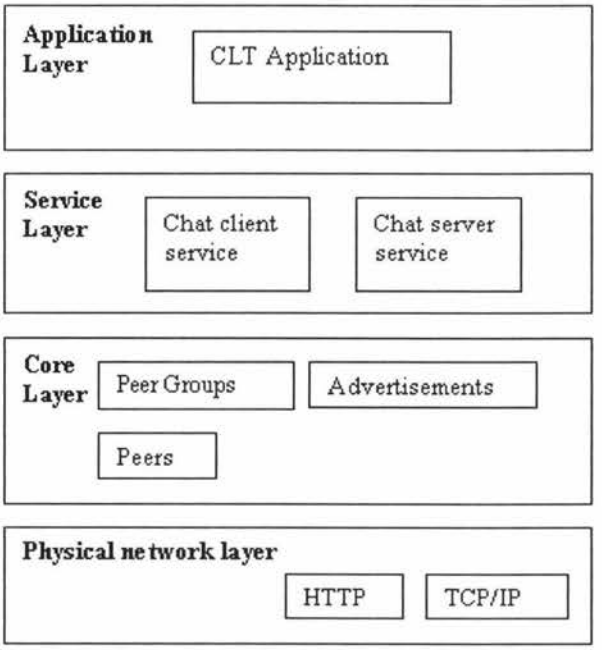


Figure 5.2 software architecture of Cooperative learning tool.

The Core Layer

The core layer of CLT is derived directly from that of JXTA. The core layer provides the elements that are essential to P2P networking.

The Service Layer

Apart from the default service which the JXTA platform is built in, there are two new services, namely chat server service and chat client service, which are built into the service layer.

The chat server service is used to accept peer's register, store peer's pipe advertisement, send list of group member to the peer and send the peer's pipe advertisement to the peer as the peer request.

The chat client service is used to register to a chat server, find the pipe advertisement and send chat messages.

The Application Layer

The CLT application is built in this layer. The CLT application includes the user interface object and the JXTA operation object. The user interface object provides the graphic user interface, which the user can interact with this application. The JXTA operation object invokes the service, which resides in the service layer, to perform CLT's functionality.

5.2.1 Networking prototype of the Instant Message

Figure 5.2.1 shows the networking prototype of the Instant Message of the CLT. The chat server creates a pipe, advertises the pipe into the group and the server listens to the pipe. The server can understand three kinds of messages: 'register message', 'finder message' and 'who is online message'. The 'register message' contains the student ID of the registering student as well as the advertisement for the pipe, which the client is listening to. When the server receives a registration message, it stores the pipe advertisement into a buffer. When the server receives the "finder message", it responds with the pipe advertisement for the other client node. With that information, the two clients can talk to each other directly. The chat server is implemented by the chat server service, the chat client is implemented by the chat client service and those messages are implemented by the TypedMessage object.

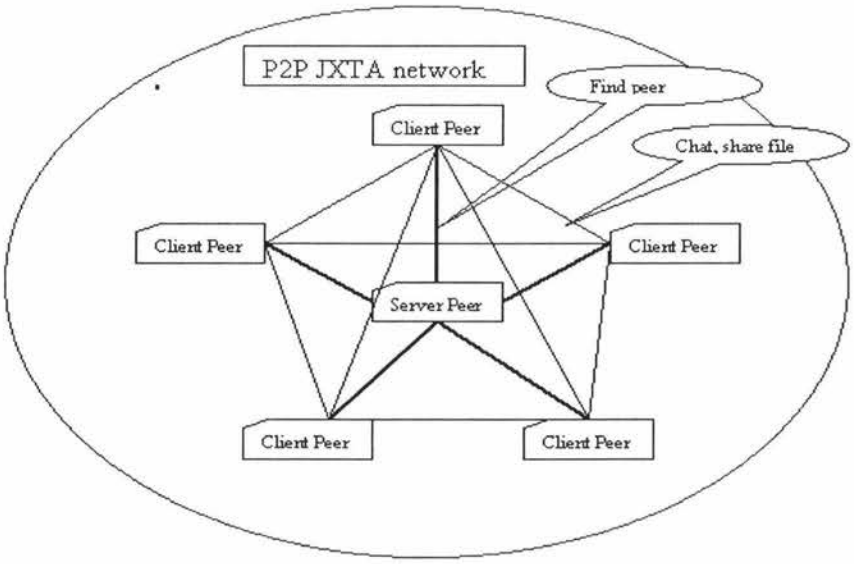


Figure 5.2.1 Networking prototype of the Instant Message

The System like Figure 5.2.1 also is called the brokered system. The Napster [17] and ICQ [18] are a similar kind of system. The real advantage of JXTA in this kind of brokered system is the firewall traversal capabilities. If both peers are behind firewalls, then in a typical peer-to-peer system they are not able to open a connection to each other. JXTA makes this firewall negotiation happen automatically, allowing system designers to write applications as if two peers are talking to each other directly even if they are behind firewalls.

5.2.2 CTL operation overview

A running CTL application represents one peer in the JXTA virtual network[19]. That peer is initially a member of the NetPeerGroup (see Section 3.3.4); all JXTA peers use this group to bootstrap to the internet or intranet. Afterwards, the peer gets the group information from Jboss application server by invoking the getGroupInfoByStudent method of the GroupControllerEJB session bean, which is exposed as web service. The group information, which was assigned by the lecturer, includes the group name and peer group ID of this peer. After the peer

get the group information, it uses the discovery service, which the JXTA platform provides, to find out if any group member in same group is on line. If the group member is found, the peer can join this group, start the chat client service, register to it with pipe advertisement and student ID. If the peer can not find any group member, it creates the peer group with name and peer group ID, which get from Jboss application server. It publishes the group advertisement, which another group member can discovery it and start the chat server service and the chat client service to become chat server in this peer group.

Each 5 minutes interval, the chat server peer invokes the createReport method of the ReportControllerEJB session bean, which is exposed as web service, to send report to the Jboss application server. The report includes the information about who is off line. Figure 5.2.2 shows CTL operation.

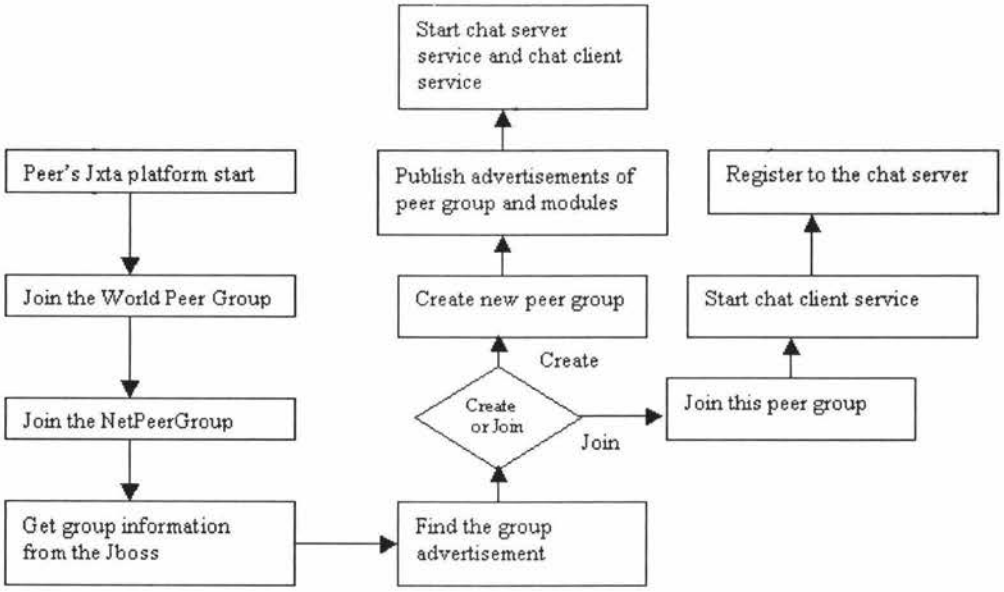


Figure 5.2.2 CLT operation

5.2.3 Cooperative Learning Tool (CLT) implementation

All CLT classes belong to the com.chatapp.chatservice package. Figure 5.2.3 shows the basic components and interaction between components.

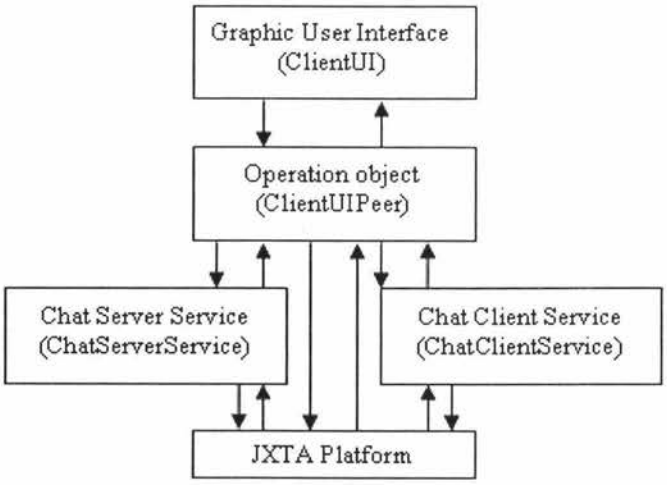


Figure 5.2.3 CLT's components interaction.

5.2.3.1 TypedMessage

The TypedMessage object is an abstraction of typed messages. This is an alterative to the JXTA Message interface. A typed message consists of two things: a "type" and a set of keyword/value pairs. This class encapsulates a simple API for manipulating typed messages, as well as methods for converting them to and from JXTA Messages. This class is not a complete replacement for the standard JXTA Message type. This class does not allow two bits of data for the same tag. It's also biased towards sending relatively small amounts of data in a message.

Constructor

The Constructor method takes the JXTA message as argument and generates the TypedMessage object.

makeJXTAMessage

Turn our TypedMessage into a JXTA message.

setValue

set the value and keyword into the HashTable.

getType

Return the message type.

getValue

According the keyword, the getValue method retrieves the value from HashTable and returns it.

There are 7 types of JXTA message, which the TypedMessage generates:

1. Register message

   A peer uses 'register' message to register to a chat server. The message consists of the type element, my ID element, and my pipe advertisements element. The pipe is an abstract communication channel built on TCP/IP or HTTP. Figure 5.2.3.1 is an example of register JXTA message.

```
<type>
  Peer Register
</type>
<My StudentID>
  00088072
</ MyStudentID>
<InputPipe>
   <jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
     <Id>
        urn:jxta:uuid-
59616261646162614...032503393B5C2F6CA7A41FBB0F890173088E79404
     </Id>
     <Type>
        JxtaUnicast
     </Type>
```

Figure 5.2.3.1 a register JXTA message

2. Finder message

   the peer uses the 'finder' message to find out another peer's pipe advertisement from the chat server. It consists of the type element, another peer's id element and my ID element.

3. Return find message

   the chat server sends the 'return find' message to the peer about another peer's pipe advertisement. It consists of a type element, peer's id element and pipe advertisement element.

4. Return not find message

   if the chat server can not find the pipe advertisement, which the peer wants to know, the chat server sends 'return not find' message to the peer.

5. Check who is on line message

   the peer uses 'check who is online' message to find out the member list of the peer group. It consists of the type element and my peer id element.

6. Return who is online message

   if the chat server sends the 'return who is online' message to peer regarding the name list of the peer group. It consists of the type element, the list of group member element.

7. Chat message

   the peer uses the 'chat' message to send chat message to another peer. It consists of the type element, the message content element and my ID element.

5.2.3.2 Chat Server Service

The Chat Server Service performs the chat server functionalities. The Chat Server Service is implemented by *ChatServerService* object. The

ChatServerService object implements the PipeMsgListener interface and Service interface which resides in the jxta.jar package.

The Service interface is identical, providing the init, startApp, stopApp, getImplAdvertisement and getInterface methods. As their names suggest, these methods are used to initialize, start, and stop the service.

The PipeMsgListener provides the pipeMsgEvent method, which monitor any event that happens in the input pipe.

myHash field

The myHash is an instance of the HashTable object. It stores a set of student ID / pipe advertisement pairs.

init

The init method implementation simply stores the passed parameters for later use. There are three arguments, which are peer group, id, and advertisement. A given peer group will be used to obtain access to pipe service of this peer group. The pipe service is bound to the peer group instance. The scope of this service resides within the group to which it is associated. This means that only peers that are active members of the group will use this pipe service to communicate to the chat server.

startApp

It creates a pipe service from the peer group instance, which is assigned in init method.

createInputPipe

The *createInputPipe* method uses pipe service, which is created in *startApp* method, to create an input pipe. This input pipe is used to accept the chat client 'register message' and 'find messages'. If any request comes, the *pipeMsgEvent* method will be trigged.

pipeMsgEvent

This method is called asynchronously whenever a pipe event occurs on the input pipe, which creates in *createInputPipe* method. This method is passed one argument: PipeMsgEvent — the event that occurred on the pipe. This method first calls PipeMsgEvent.getMessage() to retrieve the message associated with the event. After the message was retrieved, the *pipMsgEvent* constructs the *TypedMessage* object by using this message. The *pipeMsgEvent* calls private *prosessingMessage* method to process the message.

prosessingMessage

The *processingMessage* method processes message, which is passed as argument. It uses the *getType* method of the *TypedMessage* object to retrieve the type from the message.

If this message's type is 'Register message', the *processingMessage* method retrieves the student ID and the pipe advertisement from the message. It checks this student ID in *myHash*, if the student ID already exists, it updates the pipe advertisement, which associates with the student ID in the HashTable. If the student ID does not exist, it simply adds this student ID and pipe advertisement as a pair into *myHash*.

If this message's type is 'Finder message', the *processingMessage* method retrieves the student ID and the ID, which the peer wants to find, from the message. It checks the ID from the myHash. If the ID is found, it gets the associate pipe advertisement from myHash and builds this pipe advertisement into 'Return find message'. It calls the *createOutPipe* method to create an output pipe and sends this message to the peer. If the ID is not found in the myHash, it creates a 'Return not find message' and creates an output pipe to send message to the peer.

If this message's type is 'Check who is on line message', the *processingMessage* method retrieves all student ID from myHash and adds into a string. It builds this string into a 'Return who is online message'. It calls the *createOutPipe* method to create an output pipe and send this message to the peer.

createOutPipe

According to the pipe advertisement, which is stored in the myHash field, the createOutPipe method creates an output pipe. The *processingMessage* method can use the output pipe to send message to the peer.

When a peer becomes a chat server and chat client, instead of sending request to chat server, the following methods of the ChatServerService object will be used.

findClientPipe

According to the student ID, it returns the pipe advertisement.

loadMember

Returns list of the group member.

setOurClientPipe

The setOurClientPipe method sets the pipe advertisement, which associates with the chat client service into the myHash field.

5.2.3.3 Chat Client Service

The Chat Client Service performs the chat client functionalities. The Chat Client Service is implemented by *ChatClientrService* object. The *ChatClientService* object implements the *Service* interface which resides in the jxta.jar package.

myHash field

The myHash is an instance of the HashTable object. It stores a set of student ID / pipe advertisement pairs, which the peer has been received from chat server.

init

The init method implementation simply stores the passed parameters for the later use. There are three arguments, which are peer group, id, and advertisement. A given peer group will be used to obtain access to this pipe service of this peer group.

startApp

It creates a pipe service from the peer group instance, which is assigned in the *init* method.

isPipeAdv

This method is used to check if a given peer's pipe advertisement is available.

addPipeAdvtoHash

The *addPipeAdvToHash* method is used to add the student ID / pipe advertisement, which has been received from the chat server, into the myHash field.

setChatServerPipeAdv

It sets the pipe advertisement of the chat server into the local variable.

setChatClientPipeAdv

It sets this peer's student id and pipe advertisement into local variable.

registeToServer

The *registeToServer* method is used to send the 'register message' to the chat server. It builds my pipe advertisement and my student ID into the 'register message'. It creates the output pipe by using the pipe service of this group and the pipe advertisement of the chat server, which is created in the *setChatServerPipeAdv* method. It sends 'register message' to the chat server by using this output pipe.

checkWhoOnline

The *checkWhoOnlime* method is used to send the 'check who online message' to the chat server. It builds my student ID into the 'check who online message'. It creates the output pipe and sends the 'check who online message' to the chat server.

sendMessageToPeer

The *sendMessageToPeer* method is used to send a message to peer in the same group. It builds the message' content and my student ID into the 'chat message'. It retrieves the peer's pipe advertisement from myHash. It creates the output pipe by using the peer's pipe advertisement and the pipe service of this peer group. It sends the messages to the peer.

5.2.3.4 Operation Object

The Operation object performs the following functionalities:
Gets group information of the peer from the Jboss application server.

1. Creates or joins the peer group.
2. Finds out the peer, which belongs to same peer group.
3. Finds out the chat server.
4. Chat to the peer, which belongs to same peer group.
5. Sends the report of the group status to the Jboss application server.

The Operation object is implemented by *ClientUIPeer* object. A peer is represented by a unique instance of *ClientUIPeer* class. The ClientUIPeer object uses the chat server service, chat client service and the JXTA core service to perform those functionalities.

Constructor

The constructor receives an instance of *ClientUI* (see next chapter) class and assigns it to its private instance variable so the *ClientUIPeer* object can update the user interface as needed. The constructor calls the *processing* method to create the inner classes to listen to and handle the action events. It calls the *start_jxta* method to start the JXTA platform.

start_jxta

The start_jxta method instantiates the JXTA platform and creates the default net peer group, and then calls the *getPeerName* method of the net peer group to get the peer's student ID and assigns it to private instance variable. It calls the *getGroupInfo* method to get the peer's group information from jboss application server. It invokes the getService method.

getGroupInfo

The *getGroupInfor* method uses the JAXRPC Call interface to invoke the service dynamically, rather than using stub code compiled from the WSDL. It specifies the *getGroupInfoByStudent* method of the GroupControllerEJB session bean to be invoked, using the *setOperationName* method of the Call interface. The *getGroupInfo* method uses the *registerTypeMapping* method of the Call interface to define how the returned object should be handled. It calls the invoke method of the Call interface to get the GroupData object. It assigns the group name and group ID, which are wraped in the GroupData object, to class's private instance variable.

getService

The *getService* method extracts the discovery service and pipe service from the peer group. This discovery service will be used later to add a DiscoveryListener for *DiscoveryResponse* events and to send *DiscoveryRequest* messages. It calls the *joinGroup* method to join or create a group.

joinGroup

The *joinGroup* method creates a new inner anonymous class- *DiscoveryListener* object, which handles all of the incoming advertisements from a given query.

After the instantiation of the *DiscoveryLister*, the *discoveryEvent* method is created. The *discoveryEvent* method will be called each time and a new discovery response is received.

Next, the joinGroup method sends out DiscoveryRequest messages via the *getRemoteAdvertisements* method to find out the peer within the same group.

```
myDiscovery.getRemoteAdvertisements(null,DiscoveryService.GROUP,
        searchKey, GroupName, 1, myDiscoveryListener);
```

After the discovery Request message is sent, the joinGroup method is holding the thread for 16 seconds to wait for the *DescoveryResponse* message.

If a given group is not found in the discovery response message, it means that this peer is the first group member online. The joinGroup method calls the *publish* method of the DiscoveryService object to publish the *ModuleSpecAdvertisement*, which the chat server pipe advertisement builds into it. It calls the *newGroup* method to create the new group with the group name and group ID, which has been got in the *getGroupInfo* method. It calls the *startServer* method to start the chat server.

If a given group is found in the discovery response message, it means that the chat server of this peer group is available online. The joinGroup method calls the *joinThisGroup* method to join this peer group. It calls the *findAdv* method to find out the Chat server pipe advertisement.

Figure 5.2.3.4 shows the functions interaction of ClientUIPeer object.

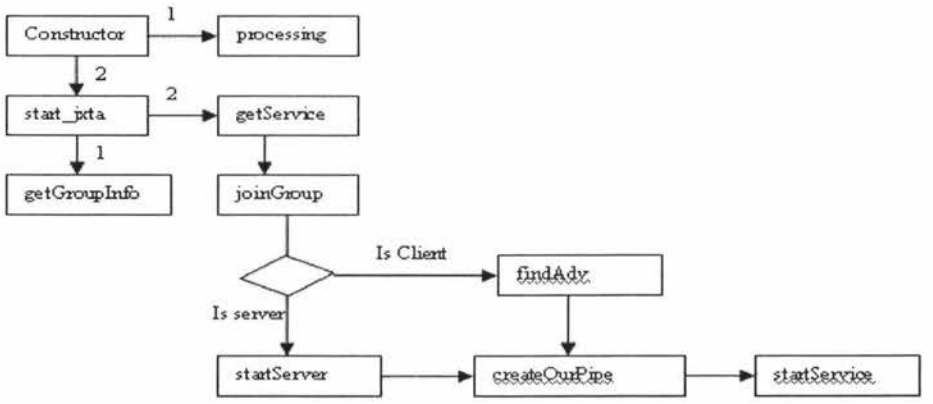


Figure 5.2.3.4 Function interaction of the ClientUIPeer

startServer

The startServer method calls the *lookupService* method of the peer group object to load the ChatServerService class. It calls the *createInputPipe* method of the ChatServerService to assign the pipe advertisement of the chat server into the ChatServerService object. This pipe advertisement is published in the *joinGroup* methods. It calls the *createOurPipe* method to create an input pipe to accept the 'chat message'.

createOurPipe

The createOurPipe method is used to create an input pipe to accept the 'chat message' or response message of the chat server. This input pipe is for the chat client service.

The createOurPipe method creates a new PipeMsgListener object as an anonymous class. After the instantiation of the *PipeMsgListener*, the *pipeMsgEvent* method is created. The *pipeMsgEvent* method monitors any event happens in this input pipe.

After the PipeMsgListener is created, the createOurPipe method calls *createInputPipe* method of the Pipe service of this peer group to create the input pipe and bind pipe listener (PipeMsgListener) to it. The createOurPIpe method calls the *startService* method to start the chat client service.

pipeMsgEvent of the Client Pipe –(inner class of the PipeMsgListener)

The *pipeMsgEvent* method check type of the incoming message first. If the message is 'return find message', it calls the *addPipeAdvtoHash* method of the ChatClientService object to store the peer's pipe advertisement into chat client service. It calls the *notifyRequester* method to wake up the sleeping thread.

If the message is 'Chat Message', it displays the 'chat message' on user's interface by calling the *append* method of the JFrame object.

If the message is 'return who is online', it calls *addIntoMemList* and *loadMemberList* methods to display the member list on the user's interface.

findAdv

The chat client peer uses this method to find out the input pipe of the chat server service.

After the input pipe of chat server service is found, the client peer can use it to send the 'register message' and 'find message'.

The findAdv method creates a new inner anonymous class- *DiscoveryListener* object, which handles all of the incoming advertisements from a given query. After the instantiation of the *DiscoveryLister*, the *discoveryEvent* method is created. The *discoveryEvent* method will be called each time and a new discovery response is received.

The findAdv method then send out discovery message to find out the *ModuleSpcAdvertisement*, which the chat server peer is published. The pipe advertisement of the chat server is created in the *ModuleSpcAdvertisement*.

If the MouduleSpeAdvertisement is found in the response message, the discoveryEvent method will be trigged and the *startService* method will be called.

startService

The *startService* method is used to start the chat client service for both the chat client peer and the chat server peer.

The startService method calls the *lookupService* method of the peer group object to load the ChatClientService object.

If this peer is a chat server, the startService method calls the *setOurClientPipe* method of the ChatServerService object to set the pipe advertisement of the chat client service into the ChatServerService object. The *startService* calls the *setVisible* of the JFrame object to display the graphic user interface and calls the *createTimerSchudule* method to send group's report to the Jboss application server in 5 minutes interval.

If this peer is a chat client, the startService method calls the *setChatServerPipeAdv* of the ChatClientService object to set pipe advertisement of the chat server, which is found before, into the ChatClientService object. It calls the *setChatClientPipeAdv* method of the ChatClientService to set pipe advertisement of the chat client into ChatClientService object. It calls the *registeToServer* method of the ChatClientService to register with the chat server. It calls setVisible of the JFrame to display the graphic user interface.

createTimerSchudule

This method creates a new java.util Timer object. It calls the *schedule* of the Timer object to set a time interval for the *run* method of the *TaskCall* object. Every 5 minutes the run method will be trigged by the *schedule* method.

run of the TaskCall object

The run method calls the *loadMember* method of the ChatServerService object to load the current online group member list. It uses this list to create a new *OnlineData* object.

The *run* method uses the JAXRPC Call interface to invoke the service dynamically rather than using stub code compiled from the WSDL. It specifies the *createReport* method of the ReportControllerEJB session bean to be invoked, using the *setOperationName* method of the Call interface. The *run* method uses the *registerTypeMapping* method of the Call interface to define how the sending object should be handled. It calls the invoke method of the Call interface to send *OnlineData* object.

processing

The *ClientUIPeer* class uses the inner classes to handle button press events. These buttons are defined in the *ClientUI* class.

After the 'Check who is online button' clicked, the *actionPerformed* method of this button listener object triggers following actions:

If it is the chat server peer, the *actionPerformed* method calls the *loadMember* of the ChatServerService object to load member list from the CahtServerService. It

calls the *addIntoMemList* and *loadMemberList* methods to display this list into user's interface.

If it is the chat client peer, the *actionPerfomed* method calls *checkWhoOnline* method of the *ChatClientService* object to send 'check who is online message' to send out 'check who is online message'. Figure 5.2.1.5 shows above operation.

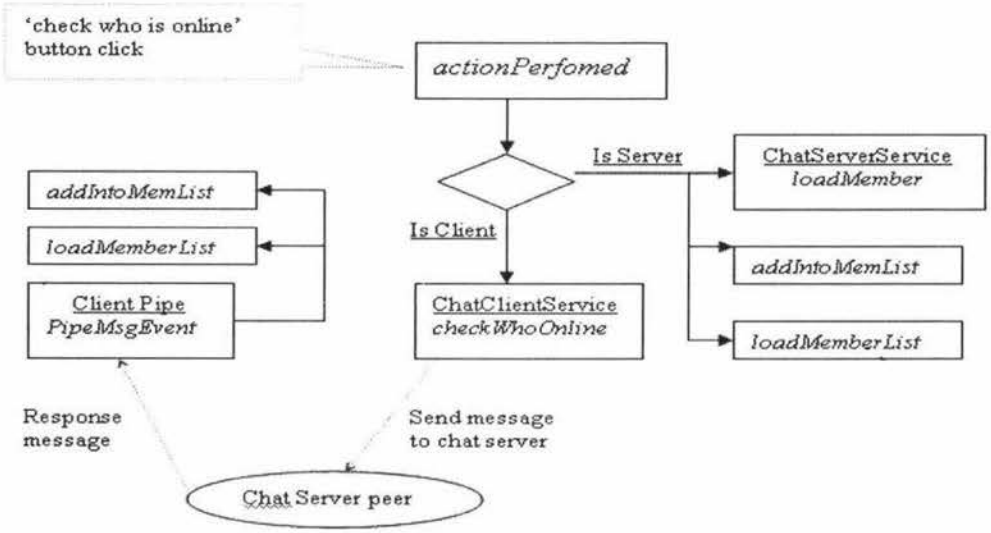


Figure 5.1.2.5 check's button click events

After the 'send message button' clicked, the *actionPerformed* method of this button listener object triggers following actions:

If it is chat server peer, the *actionPerformed* method calls *findClientPipe* method of the ChatServerService object to get the peer's pipe advertisement. It calls *addPipeAdvtoHash* method of the ChatClientService object to store this pipe advertisement. It calls *sendMessageToPeer* method of the ChatClientService object to send the message to the peer.

If it is chat client peer, the actionPerformed method calls synchronized method – *askPeerPipeAdv* to find the peer's pipe advertisement from the chat server. It calls

the *sendMessageToPeer* method of the ChatClientService object to send the chat message to peer. Figure 5.2.3.6 shows above operation.

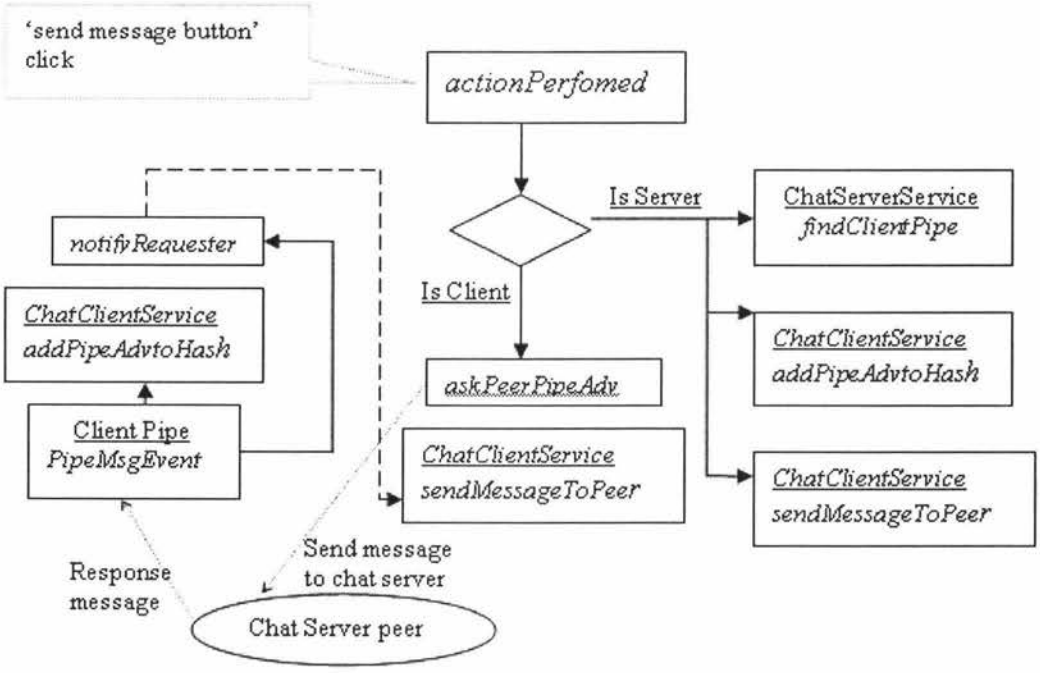


Figure 5.2.3.6 'Send message' button click events

5.2.3.5 User interface object

The User interface object provides the graphic user interface to the peer. It is implemented by the ClientUI class. Figure 5.2.1.7 shows the graphic user interface for the CLT application.

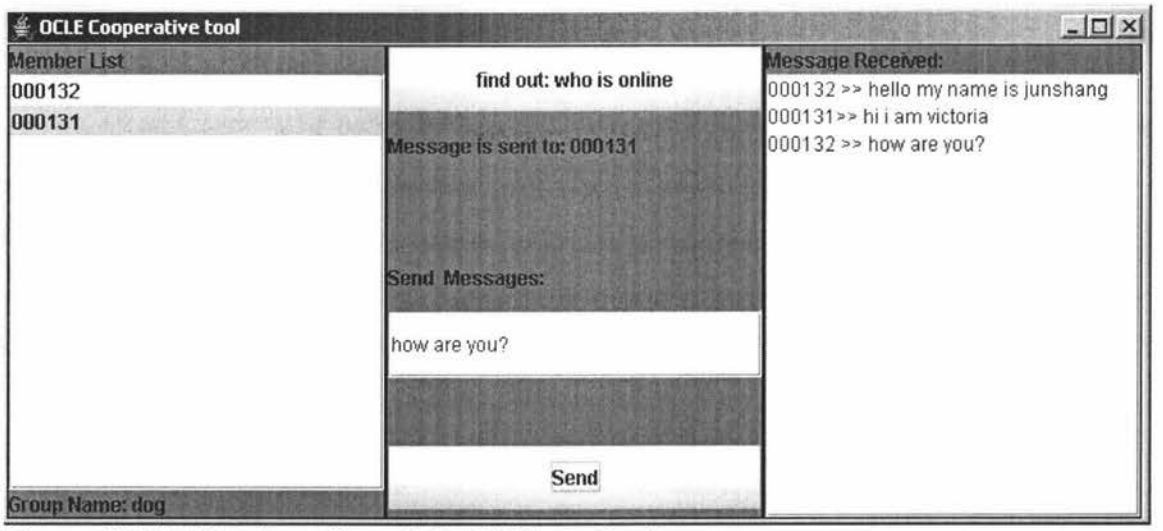


Figure 5.2.1.7 User interface of the CLT application

The *ClientUI* class, which creates the user interface, is the class with the main method and the Contractor method. The *ClientUI* extends *JFrame* object. The ClientUI object is the entry pointer of the CLT application.

Main

The main method creates instances of the ClientUI and ClientUIPeer classes. It adds a windows listener object to JFrame object.

Constructor

The ClientUI contructor creates the initial user interface, which consists of three panels. The left panel contains a JList object, which displays the member list of the group. The central panel contains a JTextField, which the peer can entry the chat message, and two JButton's, which the peer can trig the ClientUIPeer's functionalities. The right panel contains the *JTextArea*, which the received message can be displayed.

6 TEST

6.1 P2P Communication Module Compare Test

There are three communication modules, when peer communicates to each other.

- Centralized Systems: where every peer connects to a server which coordinates and manages communication.
- Brokered systems: where peers connect to a server in order to discover other peers, but they manage the communication by themselves.
- Decentralized Systems: where peers run independently without the need for centralized service.

The Brokered system is adopted by this application. The test processing compares the message passing performance of this application with Centralized system.

6.1.1 Centralized System.

In order to test processing, the centralized system is created.
Instead of sending the message to the group member directly, the peer sends the message to the chat server peer and the chat server delivers this message to the group member.

The ChatServerService and the ChatClientService objects are changed slightly.

6.1.1.1 ChatClientService

The *sendMessageToPeer* method of the ChatClientService object is changed. Instead of using the client's pipe advertisement to send a message, it uses the pipe advertisement of the chat server to send a message. This means that the message is sent to the chat server.

6.1.1.2 ChatServerService

The *processingMessage* method of the ChatServerService object is changed. After detecting the 'chat message', the *processingMessage* method finds the pipe advertisement of the group member and sends this message to the group member by using this pipe advertisement.

6.1.2 The message passing performance testing applications

Two applications are used to compare the performance of the Brokered system and the Centralized System.

Both the Brokered system and the Centralized system have the sending peer and receiving peer. The only different is: the peer of the brokered system loads the ChatServerService and ChatClientService objects from the CLT application (see Section 5.2.3.2 and 5.2.1.3)and the peers of the Centralized system loads the ChatServerService and ChatClientService from the changing ChatServerService and ChatClientServer (see Section 5.2).

6.1.2.1 Test Case 1 – two peers

After joining a peer group, the sending peer sends a message to the receiving peer. After the message is received, the receiving peer sends the original messages back to the sending peer. Each time, the back message is received by the sending peer, the time is recorded. This test processing iterates 50 times through a loop.

6.1.2.2 Test Case 2 – ring test

There are one sending peer and four receiving peers. The sending peer sends a message to the receiving peer1. After the receiving peer1 receives the message, it sends this message to the receiving peer2, which in turn sends the message to the

following receiving peer. Finally, this message is received by the sending peer. Each time, the message come back to the sending peer, the spending time is recorded. This test processing iterates 25 times. Figure 6.1.2.2 shows the ring test operation.

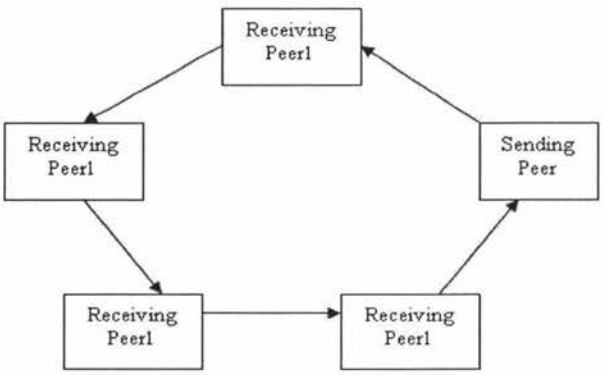


Figure 6.1.2.2 ring test

6.1.3 Testing result

6.1.3.1 Testing Environment

The following tests are done to determine the message passing performance of the brokered system and the centralized system. The testing processing is run on the local intranet, it means that there are no firewall problems. The network is in a stable state and there are no peers joining or leaving the network during the tests.

For test case 1, the sending peer and receiving peer are on different workstation.

For test case 2, there are three workstations. The sending peer is on the workstation 1, the receiving peer 1 and 2 are workstation 2 and the receiving peer 3 and 4 are on the workstation 3. Figure 6.1.3.1 shows the test environment.

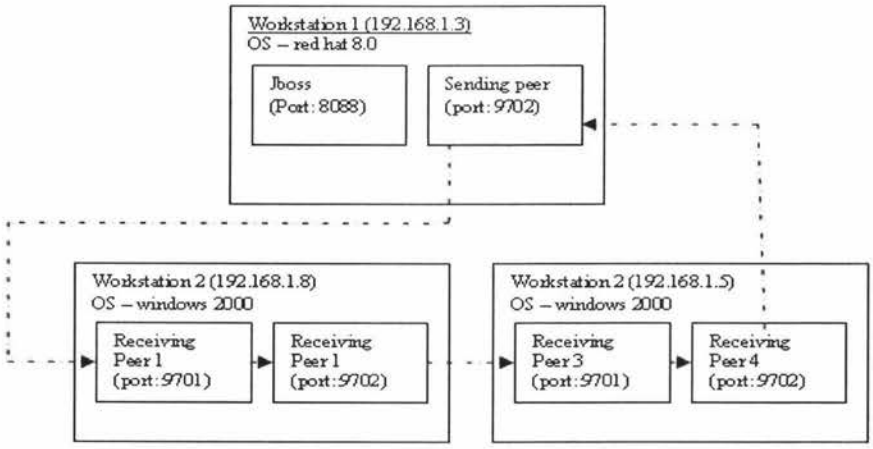


Figure 6.1.3.1 testing environment

6.1.4 Testing Result

Figure 6.1.4 shows the testing result for test case 1. The test case 1 was completed with a message looping 50 times between the sending peer and the receiving peer. The average time and total time of the brokered system is 50 % faster than the centralized system. In the first and second looping time, the brokered system is slower than the centralized system, because the peer in the brokered system needs to request the group member's pipe advertisement first and then sends the message to the group member.

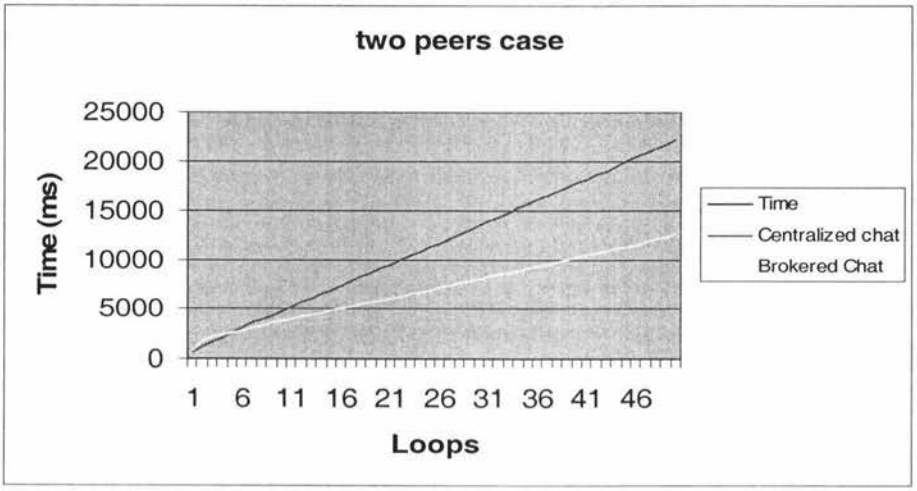


Figure 6.1.4 Testing result for two peers

Figure 6.1.4-1 shows the testing result for test case 2. The result of test case 2 is almost the same as the test case 1. The average time and total time of the brokered system is much betters than the Centralized system.

When the chat server is under heavy loads the brokered system will perform even better than the centralized system.

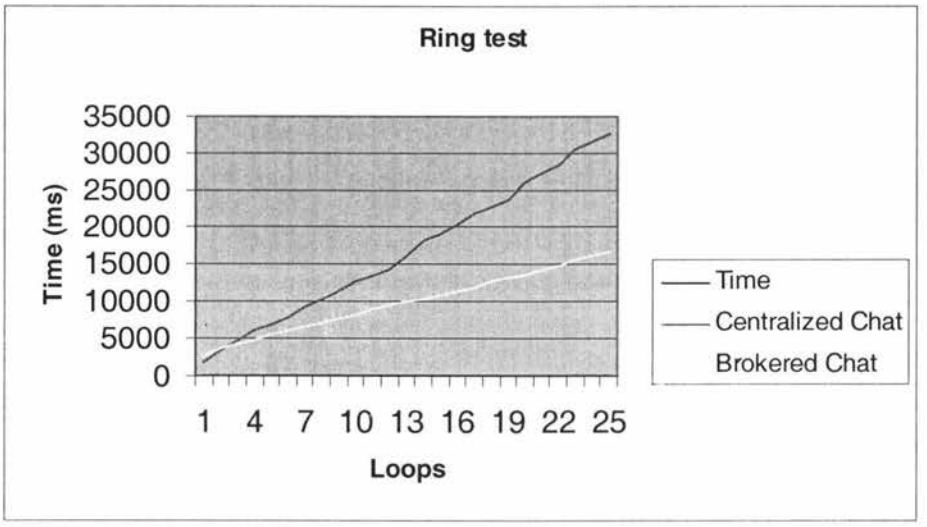


Figure 6.1.4 -1 Testing result for 25 peers

6.2 Usability Test

In order to test the user interface of the OCLE cooperative tool and the whole OCLE application's operation, a simple usability test has been developed.

6.2.1 Test Environment

In order to avoid the firewall problem, the Usability test has been done in a small computer lab which has an intranet network.

The five computers, which have been installed the windows 2000 operation system, are assigned as the client machines. Those client machines have been

installed the OCLE cooperative tool (the application of the cooperative tool is included in the CD).

The computer, which has been installed the Redhat linux 8.0 operation system and the Jboss 3.2.5 application server, is assigned as the server. The OCLE controlling component has been deployed into the Jboss application server on this machine.

The user name, password and the group have been assigned by using the OCLE controlling component (see Figure 5.1.4.1-2).

6.2.2 Representative Sample of Users

Five participants have been recruited. Those participants are university student and have used IM product (including the MSN, ICQ) regularly.

6.2.3 The Test

The five participants have been asked to do five tasks as follows:

- Start the OCLE cooperative tool and login by using password and user name, which are given by the testing administrator.

- Find a group and a group member.

- Send massage to a group member.

- After a massage has been received, send an acknowledgement message.

After finishing all tasks, the participant needed to answer the interview questions.

6.2.4 Test Result

Table 6.2.4 shows the result of the usability test for the OCLE application.

Comment type	Technical Errors	Disasters	Errors	Unfriendly	Total
No. of comments	4	0	1	2	7

Many recommendations have been garnered from the usability test. Two major trends have been identified:

- The OCLE cooperative tools should give information automatically for "a member joins"

- The OCLE cooperative tools should give information automatically for "a member leaves".

According to these recommendations, the problem in the OCLE cooperative tool has been exposed. There is no functionality, which can detect attendance of the group member automatically. This problem can be solved by adding more message type into the OCLE cooperative tool component (more details see section 7.1).

7 FURTHER WORK

7.1 The group member change dynamically

7.1.1 Chat server Change

After the peer group is formed, if the chat server peer leaves, the whole group communication will fail. The client peer can not find the pipe advertisement to send the message to the group member.

Before the chat server peer leaves, the chat server peer chooses a chat client peer randomly from its cache and sends a message to this peer "I will leave". After this chat client peer receives the message, it loads the chat server service and sends a message to rest of the group member to tell them "I am the chat server now".

7.1.2 Chat Client Change

After the peer group is formed, if any chat client peer leave, the chat server does not know the change. If the chat client peer requests the pipe advertisement of the leaving peer, the chat server still sends it to it.

Another type of message should be implemented to conquer this problem –the leaving message. Before the chat client leave, it should send a 'leaving message' to the chat server. After receiving this message, the chat server updates its HashTable and sends a message to the chat client to update the chat client's HashTable.

7.2 CLT application security issue

The peer in the CLT application uses the pipe to find a group member from the chat server and chat to a group member. The pipe in the CLT application should be secure to protect from malicious user programs.

In order to create the secure pipe communication in the peer group, the secure JXTA pipes should be used in this application. The secure JXTA pipe is implemented in the JXTA security Toolkit, which ensures that all communication between peers will be encrypted. It uses a form of SSL/TLS [20] encryption, this approach provides a transparent secure connection.

7.3 Class Domain Chat

The peer's communication in the CLT application is restricted to the study group, which the lecturer assigns. The peer can not chat with the peer in a different peer group and the lecturer can not join any peer group to chat with the student.

In order to perform the class domain chat functionalities, the large group – class domain group should be created and a new chat service should be created in the service layer of the JXTA platform. Each peer joins the class domain group first and then joins to the subgroup, which it belongs to. The lecturer peer is as super peer, which can join and chat in any peer group. The lecturer peer can broadcast the chat message to every peer in class domain. The student peer also can broadcast the message to every peer in the class domain under the lecturer's permission. Figure 7.3 shows the class domain chat.

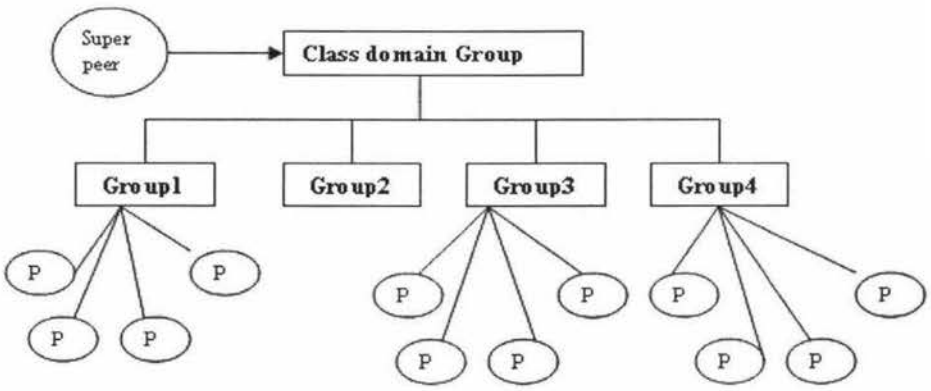


Figure 7.3 Class domain Chat

7.4 Web service security

The CLT application gets the group information and sends reports by invoking the web service of the Jboss application server. The web service is exposed and there is not any security to protect it.

Authenticating the caller method should be used to protect the web service of the Jboss application server [19]. The JBoss.net module (AXIS), which is used as the web service frame in the OCLE application, does not implement this method. So the following method may be used to protect the web service.

1. Uses XML signatures via a sister project.
2. Using HTTPs to validate the call request at the transport level.
3. Clients can authenticate themselves with the client certificates, or HTTP basic authentication.

If any those methods are used in this application, the web service in this application will be protected.

8 CONCLUSIONS

The OCLE application uses the SOA architecture to implement an online cooperative learning environment. Using advantage of the SOA, the OCLE controlling component declares a group access controlling service at a URL and the cooperative learning tool component uses the service. The two components can update separately without affecting each other. A third party application also can use the group controlling service, which is provided by OCLE controlling component. These characteristics are very useful for the OCLE application's development and combination.

The Java 2 Platform Enterprise Edition (J2EE) is a key to implementation of the OCLE controlling component. The standard security services, which are provided by J2EE (including authentication, authorization) are used by OCLE controlling component to realize the security management. The open architecture of J2EE gives the opportunity to extend the OCLE application in order to meet some new needs put forward by an educator.

In order to reduce the network traffic and server load, the P2P JXTA framework is used in the OCLE cooperative learning tool component. There are three types of communication methods in the P2P world, which are centralized system, brokered system and distributed system. The brokered system is adopted as communication method of the cooperative learning tool component. The testing result shows that is a better choice either for response time or for network traffic.

In this application, the SOAP protocol acts as a bridge between the controlling component and the cooperative learning tool component. Owing to use the SOAP protocol, the OCLE application accomplishes a prototype - central managed and distributed communication.

Due to multiply technologies (including JSP, Servlet, EJB beans, Applet, P2P and SWT) used by this application, the framework and the application server are needed. If all of these use the commercial produce, the developing cost will become expansive. Although the open source software has the shortcoming, which is hard to learning and is unstable, but it is still adopted by the OCLE application in order to reduce the developing cost.

The simple usability test detects the problem, which exists in the OCLE cooperative tool. The problem is there is no functionality, which can detect attendance of the group member automatically. Due to time consuming, this problem will be solved on later time.

There are many new services, which can be built into this application in the future. These services include file sharing, whiteboards, text slides, question manager and video chat. Once these services are built in, the OCLE application enables the student to collaborate and learn online with classmates and the lecturer in real time through a PC and an Internet connection.

REFERENCE

1. Phil, R. (1998). *500 Tips for Open and Online Learning*. London and New York: RoutledgeFalmer.

2. David, W. J., Roger. T. J. (1984). *Circles of learning: cooperation in the classroom* . USA. Minnesota 55435: Alexandria publish.

3. WebCT, Inc. *The WebCT Vision*. Retrieved June 12 2005, from http://www.webct.com/vision

4. LAMS International. *Background to LAMS*. Retrieved July 8 2005, from http://www.lamsinternational.com/about/

5. *WebCT Campus Edition 6 - Teaching and Learning Innovations.* Retrieved December 12 2005, from http://www.webct.com/ service /ViewContent?contentID=25777052.

6. *Jboss Application Server – A simple powerful J2EE application server.* Retrieved Match 26 2005, from http://www.jboss.com/products/jbossas

7. Asbury, S., Weiner, S. (2001). *Developing Java Enterprise Application.* New York: Wiley Computer Publish.

8. Vogels, W. (2003). Web Services are not Distributed Objects: Common Misconceptions about Service Oriented Architectures. *IEEE Internet Computing, 7,6,* 59-66.

9. Chappel, D. A., Jewell, T. (2002). *Java Web Services: Using Java in Service-Oriented Architectures*. O'Reilly & Associates, Inc.

10. World Wide Web Consortium. (June 2004). *SOAP Version 1.2 Part 0: Primer*. Retrieved February 6 2005, from http://www.w3.org/TR/soap12-part0/.

11. World Wide Web Consortium. (Match 2001). *Web Services Description Language(WSDL) 1.1*. Retrieved October 09 2004, from http://xml.coverpages.org / wsdl20000929.html

12. OASIS international consortium.(2000). *UDDI Technical White Paper.* Retrieved May 15 2005, from http://www.uddi.org/whitepapers.html.

13. Steve, G., Doug, D., Simeon, S. (2005). *Building web Services with Java*. USA: Sams publishing.

14. JXTA organization. (2005). *Project JXTA: Java Programmer's Guide.* Retrieved December 12 2005, from http://www.jxta.org /docs /jxtaprogguide_final.pdf.

15. Joseph, D. (2002). *Mastering JXTA: Building Java Peer-to-Peer Applications.* New York: Wiley Computer Publish.

16. Dirk, R. (2001). *Connect the enterprise with the JCA, Part 1- A look at the J2EE Connector Architecture.* Retrieved June 28 2005, from http://www.javaworld.com/javaworld/jw-11-2001/jw-1121-jca.html.

17. Mark, D. (2001). Surfing the net for software engineering notes. *ACM SIGSOFT Software Engineering Notes, 26,4,* 17-26.

18. Bernard, T., Mohamed, A. (2002*). Project JXTA Virtual Network.* Palo Alto, CA 94303 USA.

19. The Apache Software Foundation. (2005). *Web Service security –Axis.* Retrieved November 11 2005, from http://ws.apache.org /axis /java/security.pdf.

20. Internet Engineering Task Force. *Transport Layer Secruity (TLS).* Retrieved June 17, 2005, from http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc2246-bis-02.txt.

21. BEA Systems, inc. *BEA WebLogic Server® 9.2.* Retrieved February 10, 2006, from http: //www.bea.com/ content/news_events/ white_papers /BEA_WLS_Prod_Brief_ds.pdf.

22. Harasim, L.M. (1990). *On-line education: Perspectives on a new environment.* NY: Praeger Publishing.

23. Slavin, R.E. (1995). *Cooperative learning: Theory, research, and practice (2nd).* Allyn and Bacon, Needham Heights MA.

24. O'Malley, C. (1994). *Computer-supported collaborative learning. Springer.* New York: Dover.

25. THORSTEN, H., REINHARD, K.S. (Summer 2001). sTeam: Structuring Information in Team–Distributed Knowledge Management in Cooperative Learning Environments. *ACM Journal of Educational Resources in Computing,1,* 2. p.p 27.

26. Maria, B., Giuseppe, C., Umberto, F. P., Vincenzo, G., Claudia, R., Vittorio, S., et al. (2000). Teach++: a cooperative distance learning and teaching environment. *Symposium on Applied Computing archive Proceedings of the 2000 ACM symposium on Applied computing , 1,* 124 – 130.

27. Sharon, Z., Scott, H., Jacob, R., Isaac, R., Tom, R., Mark, H. (2006). *The Java(TM) Tutorial: A Short Course on the Basics (4th).* Addison-Wesley Professional, NY.

28. Changtao, Q., Thomas, E., Christoph, M. (2000). Implementation of a WebDAV-based collaborative distance learning. *Proceedings of the 28th annual ACM SIGUCCS conference on User services: Building the future,* 258 – 265.

29. Reschke, J. F. (2005). *Datatypes for Web Distributed Authoring and Versioning (WebDAV) Properties.* Retrieved July 30 2006, from http: / www.webdav.org /specs/rfc4316.html

30. Michael, D. C., Neil, T., Michael, D. (2005). Developing a synchronous web seminar application for online learning. *Proceedings of the 33rd annual ACM SIGUCCS conference on User services,* 36 - 39.

31. JEFFREY, R. Y. (2002). *Pricing Changes by Blackboard and WebCT Cost Some Colleges More -- Much More.* Retrieved August 22 2006, from http://chronicle.com/free/2002/03/2002031901u.htm.