Rochester Institute of Technology

MS Project Proposal

GROUP DATA COMMUNICATION OF M2MI VIA LAN

-- M2MI-Based Collaborative Groupware

By Kai Cheng
kxc8217@cs.rit.edu

Version 8.0

Committee:

Chairman: Prof. Hans-Peter Bischof
Reader: Prof. Fereydoun Kazemian
Observer: Prof. Paul T. Tymann

Table of Contents

1. Abstract of Project Objectives

This graduate project will provide three different design patterns of M2MI-based collaborative systems and compares the advantages and disadvantages of the M2MI-based solutions with RMI-based solutions of those three different problems, collaborative groupware, multiple-users chat system, and the distributed solution of shared resource allocation.

Groupware Description

Appointment Making Groupware Application

It has calendar-scheduling feature, for example: appointment request, acceptance, rejection, modification, cancellation, and display, etc.
This system has the following features:

- Create an appointment of a specified date and time.
- Send an appointment the another instance of the program
- Display appointments for a given date, week, or month
- Update an existing appointment with another user
- Cancel an existing appointment

Multiple Participants Chat Applications

It has multiple participants chatting feature, for example: instant chatting between two users, chatting within a group, and recovering lost chat messages, etc.
This system has the following features:

- Instant message – instant message exchanging between two users
- Group chatting – messages exchanging within multiple users
- Chat log recovery – recovering lost chat messages

Dining Philosophers Solution

This application is designed for the distributed solution of the Dining Philosophers problem when using in ad hoc environment. It provides the distributed solution for shared resource allocation in network. The main objective of this solution is to find a reasonable design to allocate shared resource to separate users (running on different computing device, which means has different Java Virtual machine), make sure they have equal opportunity to use shared resource, and avoid deadlock.

Research Objective Concepts

This project has the following research concepts:

- Investigate the design pattern and model design of collaborative groupware
- Investigate the JAVA design and implementation of the collaborative groupware
- Investigate M2MI mechanism using in the three different problems in ad hoc environment.
- Investigate the architecture, mechanism and performance of the designs of the three problems and compare them with RMI based solution and basic M2MI implementation. Test will be performed while using varieties of M2MP packets and network bandwidths. Detect the existence and hazards of bottlenecks and investigate how to avoid them.

2. Architecture Overview

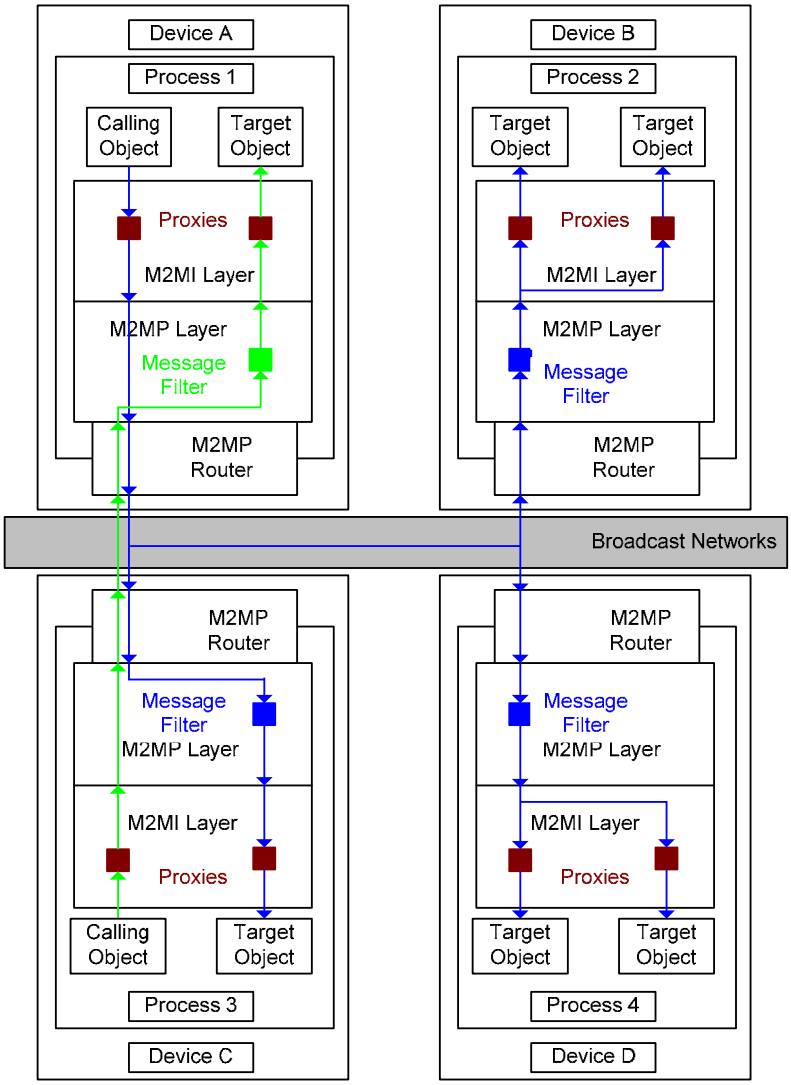


Figure 1: Collaborative Groupware System Architecture

Figure 1 shows the architecture of M2MI-based collaborative system. It is suitable for small computing device running in wireless ad hoc networks environment. When a method is invoked on an object's omnihandle for a particular interface, all the target object, which implements the same interface, will perform the same method. If a method is invoked is on a unihandle, only the object bounded to this unihandle performs the method. If a method is invoked on a multihandle, the set of objects that implement the same interface will perform the method.

3. Design Patterns

3.1 Multiple Participants Chat System

3.1.1 Background and Problem

The chat system can perform the following functions and may encounter the following problems.

Instant Chat

A user can type in texts, and those texts can be displayed on the user's device that the sender specifies. In this section, we only deal with one-to-one chat. The one-to-many chat (multiple chat sessions) will be treated in Group Chat section.
To achieve that, we may have the following problems:

- Every user must have a local active user list so it can look up this list to find the user it wants to send message to
- The messages are only displayed on the two users' devices involved in the instant message chat session. This can avoid unnecessary data transmission and broadcast storm
- The messages are displayed on the users' devices involved in the chat session in the order that they are typed in
- Every active user should keep notifying all the other active users if it is still available to receive instant message

We will discuss the solution in the design pattern section.

Group Chat

A user can create a multiple participants chat session and the users involved can add themselves respectively in the group chat session. Whenever a user in a chat group types in texts, those texts can be displayed on all the users' devices in that particular chat group.
To achieve that, we may have the following problems:

- Every user must have a local active user list so it can look up this list to find the users that it wants to invite to create a group chat session
- Every user can create a multiple participants chat session and send it to all the users involved
- A user who joins later can find the group chat session and add himself in it
- All the users in a group chat session can type in texts and the texts can be displayed on all the users' devices in that group chat session
- The messages are only displayed on the devices involved in the group chat session. This can avoid unnecessary data transmission and broadcast storm
- The messages are displayed on the users' devices involved in the chat session in the order that they are typed in
- Every active user should keep notifying all the other active users if it is still available to receive group messages

We will discuss the solution in the design pattern section.

Chat Log Recovery

When some users are out of the service area (since this collaborative chat system is designed based on M2MI mechanism and compatible with wireless proximal ad hoc networks), or messages are lost during broadcasting, all users' devices should have the ability: detecting messages lost and recovering the lost messages by synchronizing with other users' devices.

To achieve that, we may have the following problems:

- Every user can detect if it has all the messages that it should have received
- When a user finds it does not have all the messages, it can synchronize with other devices to get the lost messages recovered
- To avoid unnecessary data transmission and broadcast storm, users exchange chat logs as less as possible

We will discuss the solution in the design pattern section

3.1.2 Design Pattern

"Patterns are devices that allow programs to share knowledge about their design."[7] Generally, same problem may occur again and again. So developer has to repeat solving almost the same problem many times. Pattern is a better way that we can make the design that we have developed to solve a specific program design problem reusable.

3.1.2.1 Design based on M2MI mechanism

Pattern Name

M2MI Instant Message-exchange **(a documenting design pattern** [7]**)**

"Documenting patterns is one way that developer can reuse and possibly share the information that users have learned about how it is best to solve a specific program design problem. Documenting design patterns are usually done in a fairly well defined form. The general form for documenting patterns is to define items such as:

1. The motivation or context that this pattern applies to
2. Prerequisites that should be satisfied before deciding to use a pattern
3. A description of the program structure that the pattern will define
4. A list of the participants needed to complete a pattern
5. Consequences of using the pattern...both positive and negative
6. Examples"

Intent and Motivation

The ***M2MI Instant Message-exchange*** pattern applies to situations in which small computing devices, who run in ad hoc network, need to perform the following operations, include user identification, instant message sending , instant message receiving, group chatting, and chat log recovery, etc

Applicability (or Prerequisites)

The ***M2MI Instant Message-exchange*** pattern is used in classical ad hoc environment for multiple participants chatting, which means no central server, no network

administration, no complicated routing protocol and no complicated system development. It is suitable for small set of users, small computing devices and limited working space (Noisy or Quite space)

Solution

The following class will be developed when implement the ***M2MI Instant Message-exchange*** pattern.
Each ***M2MI Instant Message-exchange*** application exports an object implementing these following interfaces:

```
public interface UserReporter {
        // developer can add more arguments and more method
        public void declare (IME ime, String userID);
        public void request ();
        public void reply (UserRec [] ul);
        public void GroupIMCreate (IME groupIME, String chatName);
        …}
```

UserRec, which is the records set for every users in service, implements Java Serializable with **IME ime, String userID,** and **Long timestamp,** etc

Interface that is designed for instant message sending, group chatting, get the latest message ID and get the message with specified number.

```
public interface IME {
        // developer can add more arguments and more method
        public void talk (String msg, Long msgID, IME sender);
        public void mulTalk (String msg, Long msgID);
        public void msgRequest (Long msgID, IME sender);
        public void latestMsgReq ();
        public void latestMsgRep (Long msgID);
        …}
```

Participants

- Initial class, which declares a user process, a group chat session, and have the member methods for the request of active user list and the reply. The ***Instant Message-exchange*** application obtains an omnihandle from M2MI layer for this initial interface
- Main class, which implements instant message sending, group message sending, latest message request and reply, and message recovery methods, etc. The ***Instant Message-exchange*** application obtains an unihandle or multihandle from M2MI layer for this interface

3.1.2.2 Design based on RMI mechanism

Pattern Name

RMI Instant Message-exchange **(a documenting design pattern [7])**

Intent and Motivation

The ***RMI Instant Message-exchange*** pattern applies to the situations in which regular computing devices(PC, Laptop, etc), running in internet or intranet environment, need to perform the following operations, include instant message sending , instant message receiving, group chatting, and chat log recovery, etc

Applicability (or Prerequisites)

The ***RMI Instant Message-exchange*** pattern is used in internet or intranet environment for multiple participants chatting, which has a powerful central server, network administration, and complicated routing protocol. It is suitable for big set of users, regular computing devices (PC, Laptop, etc) and large working space (May cross the earth)

Solution

Every user acts as a client part for this ***RMI Instant Message-exchange*** system. Before a user starts to work, its device gets an IP and port from network administrator. Then it looks up the RMI chat server with the host IP and port.

```
public ChatClient( String host, int port ) {
        rmiServer = (Chat) Naming.lookup("rmi://" + host + ":" + port + "/ChatServer");}
```

After the connection is established, every client should apply a userID from the RMI server then it can call the methods like local method invocation.

```
public int apply(String userName, String senderHost, int port) {
        return rmiServer.userReg(username, senderHost, port);}
public String talk( String sender, String receiver, String text)
                        throws ChatException, RemoteException {
        return rmiServer.msgTrans(sender, receiver, text);}
```

In the Server side, whenever it receives a text from a user, it assigns an ID for that message and establishes a TCP connection to the receiver. After the receiver replies, the server returns the reply message to the sender. The server needs to provide an interface to declare all the functions it can provide.

```
public int userReg(String username, String clientHost, int clientPort){
        assign a unique userID for the client;
        stores the client's host and port; }
public String msgTrans(String sender, String Receiver, String text){
        looks for the sender's ID and receiver's ID in user list;
        creates a TCP connection and send the text to the receiver;
        get the reply from the receiver;}
```

Participants

- Client class, which applies a membership from RMI server, and has the member functions like chatting with single or multiple users.
- Server class, which provides instant message sending, group message sending, latest message request, and message recovery methods, etc.

- Server interface class, which provides interface for all the member functions in server class.

3.1.3 RMI-based Design vs. M2MI-based Design

Figure 2 shows the architecture of RMI-based design for this Chat system. The Chat server calls the RMI registry to associate a name with a remote object. When a user wants to chat with other users (invoking a method on the remote object), the client procedure looks up the registry, gets the remote object by its name, and calls the appropriate method on the client stub. Then the client stub builds the appropriate messages and traps to the handler, which sends messages to the Chat Server through the remote kernel. Then server unpacks the parameters and calls the procedure. The procedure communicates with other users via TCP connection and gets the reply. The server packs it in a message and transmits it to the original sender. All the end users feel like local method invocation.

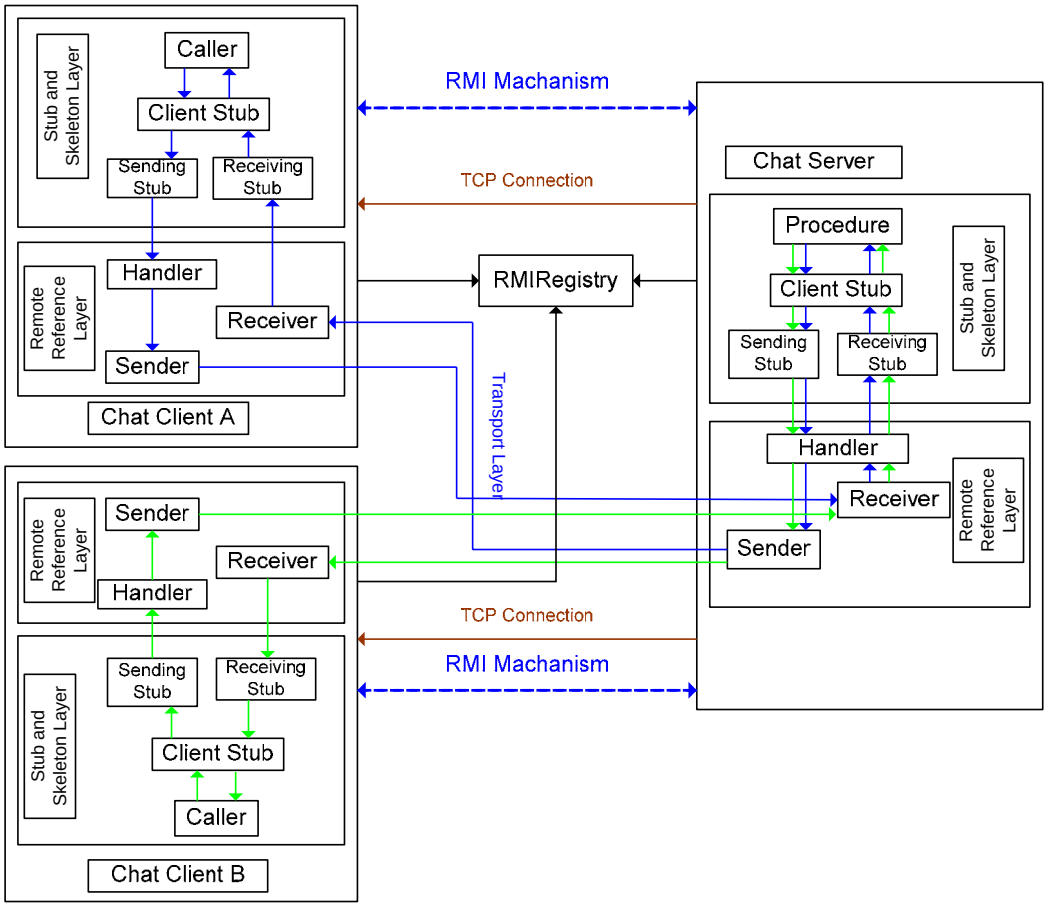
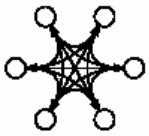
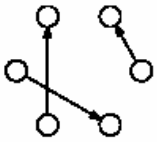


Figure 2: Architecture of RMI Design for the Multiple Participants Chat System

M2MI-based Design		RMI-based Design	
Object oriented abstraction of broadcasting communication 		Object oriented abstraction of Point-to-Point communications 	
Conclusion	**Detailed Reason**	**Conclusion**	**Detailed Reason**
Compatible for serer less distributed applications	Ad hoc network based on broadcasting	Must have a powerful server to avoid bottleneck	Code-base server must provide proxy classes downloading since client calls methods on proxy classes
All of those steps are not necessary for M2MI. Simplify system deployment	Ad hoc collaborative system characteristics	Programmer must do those steps manually and make sure all of them are correct otherwise it will fail	RMI calls methods on an object from a remote host using a remote stub. The remote stub class must be created using rmic and installed in a code-base server. The server property must be set to the remote stub class's URL under the code-base server. The remote object must be exported into a registry like the Jini Lookup Service to set up a remote object.
Save response time, computing resource and network resource	Don't need to compile the objects ahead of time and then transmit them. All the devices in this system already have the objects	Needs more response time, computing and network resource	The proxy classes must be compiled ahead of time from an actually JAVA interface and then installed to all the devices involved in this application
More secure to users	The objects are already in devices	Download codes from internet may not be secure	Server faces attacks like tremendous code download requests crash server. Third party may pretend as server. The download codes can't be fully trusted
Simpler for system development	Server and this request is not necessary	More complicated system development	Security policy must be put in place to permit connecting to the code base server
Save response time, computing	All messages are delivered by	Needs more response time,	The server must keep IP addresses and port numbers for

resource and network resource	broadcasting so we do not need IP address and the port number	computing and network resource	all the devices that in the environment
Not necessary	Based on broadcasting, don't need IP and port number	Network administration	Before system works, IP address and port number need to be assigned to identify every device
Devices can join and leave without prior notification and setup	User identification is solved at application level	Devices can't join and leave freely. They need to get approvals from server	RMI server needs a member list which is identified by IP address or other unique ID. So it can send message back to clients
M2MI-based Design		**RMI-based Design**	
"When an object of a non-primitive type is passed directly as an M2MI method call argument, the object is normally passed by copy, that means, manipulations of the argument by the method call recipient do not affect the original object in the caller"		RMI passes objects by their true type, so the behavior of those objects is not changed when they are send to another virtual machine	
M2MI uses Java's object serialization to marshal the method call arguments on the calling side and unmarshal them again on the target side		RMI marshals all reference and pack together in the message (stream), require a table of link page with the message to the locations of sub objects	
Suitable for small devices that do not have powerful CPU and battery		Need powerful computing devices, like desktop or laptop	
Don't need network routers, switches, and protocols when data transferred		Routers, switches, and network protocols are necessary because it is based on TCP connection	
Not 100% reliable – based on M2MI mechanism and broadcasting		100% reliable – RMI based on TCP protocol. Acknowledgement is provided.	
Communication can only be established in a certain small area		Communication can be established cross the Earth via internet	
Eliminating server side development since the collaborative system is server less		Simplify client side application development since developer uses remote methods implemented in server side just like local method invocation	

Figure 3: M2MI-based Design vs. RMI-based Design

3.2 Appointment Making Groupware System
3.2.1 Background and Problem

The appointment making groupware system can perform the following functions and may encounter the following problems.

Make an Appointment and Accept (or Reject) It

A user can create an appointment and send it to another user. The information of the appointment can be displayed on both users' devices. After both users accept the appointment, the data will be saved in both users' devices otherwise it will be discarded
To achieve that, we may have the following problems:

- Every user must have a local active user list so it can look up this list to find the user it wants to send appointment to
- Every user can receive appointment request and then send back the reply to accept it or reject it
- Once an appointment is accepted by two users, it will be saved in the two users' local devices otherwise it will be discarded
- The information of appointments is only displayed on the two users' devices involved in the appointment making process. This can avoid unnecessary data transmission and broadcast storm
- Every active user should keep notifying all the other active users if it is still available to receive appointment

Update (or Cancel) an Existing Appointment and Accept (or Reject) It

A user can send the update or cancel request of an existing appointment to another user involved in that appointment, and the receiver can accept or reject it. The data will be saved in both users' devices if both user accept it otherwise it will be discarded
To achieve that, we may have the following problems:

- Every user must have a local appointment list so it can find the existing appointment and the other user involved in it, and then send the updating (include canceling) request to the user
- Every user can receive updating appointment request and then send back the accepting reply or rejecting reply
- Once the update of an appointment is accepted by two users, the updated appointment will be saved in the two users' devices otherwise it will be discarded. If an appointment is canceled, it will be removed from both users' devices
- The information of appointments is only displayed on the two users' devices involved in the appointment making process
- Every active user should keep notifying all the other active users if it is still available to receive appointment

Exceptions

We need to handle the exception in the application level since the data transmission of M2MI-based collaborative groupware system is not 100% reliable. Data may be lost during broadcasting. The following problems may occur:

- While new devices come in, all users need to get an updated user list
- If the receipt is out of the broadcasting area or does not reply, the sender will wait a certain time for the reply and then try additional two times, if the receipt still not reply, then the process fails with an appropriate message
- When a sender sends an appointment with another device, if any interruption, like data transferred error, broadcasting problem, or just the receipt walks out of broadcasting area, this error occurs. The application will check the date reliability and integrity to make sure all the users get the correct data

We will discuss the solution in the design pattern section.

3.2.2 Design Pattern

3.2.2.1 Design based on M2MI mechanism

Pattern Name

M2MI Collaborative Groupware **(a documenting design pattern [7])**

Intent and Motivation

The ***M2MI Collaborative Groupware*** pattern applies to the situations in which small computing devices, running in ad hoc network, need to perform the following operations, include event creation, event message sending, event message receiving, event confirmation, event rejection, event updating, and event display, etc.

Applicability (or Prerequisites)

The ***M2MI Collaborative Groupware*** pattern is used in classical ad hoc environment for multiple participants' event making, which means no central server, no network administration, no complicated routing protocol and no complicated system development. It is suitable for small set of users, small computing devices, and limited working space.

Solution

The following class will be developed when implement the ***M2MI Collaborative Groupware*** pattern.
Each ***M2MI Collaborative Groupware*** application exports an object implementing these following interfaces:

```
public interface CG {
        // developer can add more arguments and more method
        public void declare (CGReceiver cgr, CGSender cgs, String userID);
        public void request ();
        public void reply (UserRec[] ul);
        … }
```

UserRec, which is the records set for every users in service, implements Java Serializable with **CGReceiver cgr, CGSender cgs, String userID,** and **Long timestamp,** etc.
The following interface is designed for the event acceptance, rejection, and reply.

```
public interface CGSender {
        // developer can add more arguments and more method
        public void repEvt (String reply, Long evtID);
        public void rejEvt(String reason, Long evtID);
        … }
```

The following interface is designed for the event request, modification, and cancellation.

```
public interface CGReceiver {
        // developer can add more arguments and more method
        public void getEvt (the arguments for event, Long evtID, CGSender cgs);
        public void updEvt (the arguments for event, Long evtID, CGSender cgs);
        public void canEvt (String reason, Long evtID, CGSender cgs);
        …}
```

Participants

- ***M2MI Collaborative Groupware*** initial class, which declares a user process and have the member methods for the request of active user list and the reply. The ***M2MI Collaborative Groupware*** application obtains an omnihandle from M2MI layer for this initial interface
- ***M2MI Collaborative Groupware*** event message receiving class, which implements events' messages receiving, events rejection methods, etc. The ***M2MI Collaborative Groupware*** application obtains an unihandle from M2MI layer for this interface
- ***M2MI Collaborative Groupware*** event message sending class, which implements messages sending, updating, etc. The ***Collaborative Groupware*** application obtains an unihandle from M2MI layer for this interface

3.2.2.2 Design based on RMI mechanism

Pattern Name

RMI Collaborative Groupware **(a documenting design pattern [7])**

Intent and Motivation

The ***RMI Collaborative Groupware*** pattern applies to the situations in which regular computing devices (PC, Laptop, etc), running in internet or intranet environment, need to perform the following operations, include event creation, event message sending, event message receiving, event confirmation, event rejection, event updating, and event display, etc.

Applicability (or Prerequisites)

The ***RMI Collaborative Groupware*** pattern is used for multiple participants' event making in internet or intranet environment, which has a powerful central server, network administration and complicated routing protocol. It is suitable for big set of users, regular computing devices (PC, Laptop, etc) and large working space (May cross the earth)

Solution

Every user acts as a client part for this ***RMI Collaborative Groupware*** system. Before a user starts to work, its device gets an IP and port from network administrator. Then it looks up the RMI server with the host IP and port. After the connection is established, every client should apply a userID from the RMI server then it can call the methods like local method invocation.
In the Server side, whenever it receives an appointment request from a user, it assigns an ID for that appointment and establishes a TCP connection to the receiver. After the receiver replies, the server returns the reply message to the sender. The server needs to provide an interface to declare all the functions it can provide.

Participants

- Client class, which applies a membership from RMI server, and has the member functions like event request, reply, update, and cancel.
- Server class, which provides event request, event reply, event updating, and event cancellation, etc.
- Server interface class, which provides interface for all the member functions in server class.

3.2.3 RMI-based Design vs. M2MI-based Design

Comprising to RMI-based design of appointment making system, the M2MI-based design has the following differences:

- RMI needs to send messages to users one by one by a central server, while M2MI use broadcast to send to all users at the same time.
- RMI needs a powerful code-base Server, while M2MI is server less
- RMI needs to pre-compile code, while M2MI requires every device has the classes ahead of time
- RMI needs IP and port assignment and network administration, while M2MI is designed based on ad hoc environment and does not need that
- RMI needs exactly membership, while M2MI can allow devices come and go free without notification
- RMI simplify the development for client application, while M2MI simplifies the development of application for programmer
- RMI has 100% reliable data transfer, while M2MI don’t have that.

3.3 Dining Philosophers Solution

3.3.1 Background and Problem

The dining philosophers is a classical example to illustrate various problems(like deadlock and equal opportunity for all users) that can occur when many synchronized threads are competing for limited shared resources.

“Five philosophers are sitting at a circular table. Each philosopher spends his life alternately thinking and eating. In the center of the table there is a bowl of rice. Between

each pair of philosophers is one chopstick. When hungry, each philosopher must get his left and right chopstick before he can start eating. After he is done, he puts down the chopsticks and let other philosophers eat. The philosophers must find some way to share chopsticks such that they all eat with a reasonable frequency and avoid deadlock." [6] (Figure 4)

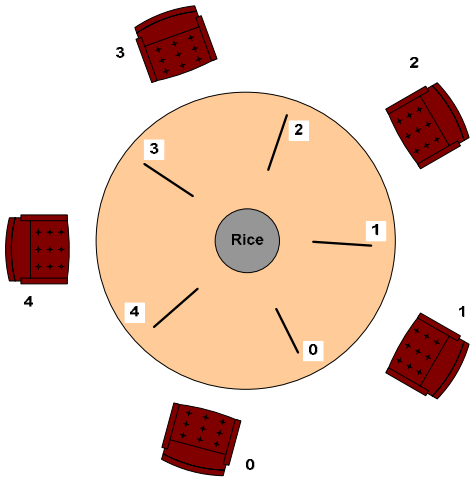


Figure 4: Dining Philosophers problem

To provide a distributed solution of Dining Philosophers, we may encounter the following problems:

- We assume the five philosophers (consumers) are already sitting around a circular table and the five chopsticks (shared resources) are already put on the table. That means, we don't handle later join-in consumers and resources.
- Every philosopher knows which chopsticks he needs
- Every chopstick can only be occupied by one philosopher each time, that means, when a shared resource is occupied, it will block all the occupy requests from other consumers
- Every Philosopher must have equal opportunity to eat rice
- Deadlock must be avoided, that means, we should avoid the situation that every philosopher is waiting for another one (circular deadlock).

We will discuss the solution in the design pattern section.

3.3.2 Design Pattern

3.3.2.1 Design based on M2MI mechanism

Pattern Name

M2MI Shared Resource Allocation **(a documenting design pattern [7])**

Intent and Motivation

The ***M2MI Shared Resource Allocation*** pattern applies to situations in which small computing devices, running in ad hoc network, need to compete for some shared

resource. Every device should have equal opportunity to use the shared resource and avoid deadlock.

Applicability (or Prerequisites)

The ***M2MI Shared Resource Allocation*** pattern is used in classical ad hoc environment for multiple devices competing for shared resource, which means no central server, no network administration, no complicated routing protocol and no complicated system development. It is suitable for fix-number users and resource, small computing devices, and limited working space.

Solution

The following class will be developed when implement the ***M2MI Shared Resource Allocation*** pattern.
Each Consumer process in the ***M2MI Shared Resource Allocation*** pattern exports an object implementing the following interface. The application obtains an omnihandle for this interface.

```
public interface ComsumerCreate {
        //developer can add more arguments and more methods
        declare (Consumer con, int consumerID, int resouceNeed1, int reourceNeed2, …);
…}
```

resourceNeed is the shared resource that the consumer needs

Each Consumer process implements the following interface. The application obtains a unihandle for this interface.

```
public interface Consumer {
        //developer can add more arguments and more methods
        checkResourceNeed (int resourceNeed1, int resourceNeed2, …);
…}
```

Each resource process exports an object implementing the following interface. The application obtains an omnihandle for this interface.

```
public interface ResouceCreate {
        //developer can add more arguments and more methods
        declare (Resource r, int rID);
        sayAva (Resource r, int rID);
        sayUse (Resource r, int rID);
…}
```

The following interface is for every shared resource to be used and released. Whenever **use** is invocated on a **Consumer** object, it blocks all the other invocations from the other consumers. The application obtains a unihandle for this interface.

```
public interface Resource {
        //developer can add more arguments and more methods
        use ();
        put ();…}
```

When every consumer sits on the table (namely, comes into service), it declares itself with **declare** method. Then every resource declares itself when it is available at the first time. Assume all consumers have already sit around the table (namely, ready for receive the resource declaration) and the data transmission is perfect, every consumer has the available resource list now (Initially, all resources are available). Then every consumer searches the available resource list by calling the **checkResourceNeed** method for his left and right **Resource** unihandles. Only if both two resources are in the available list, the consumer continues to call the **use** method on both **Resource** objects, and the two **Resource** objects call **sayUse** method on the omnihandle. Then every **Consumer** removes those **Resource** unihandles from available resource list and stops any **use** invocations on them so no other consumer can use those resources at that time. For example, if two consumers next to each other call **checkResourceNeed** at the same time, both of them will get the reply that the resources are available. But only the first consumer that calls **use** method will win while the second consumer stops the **use** invocation after it knows those resource call **sayUse** to indicate they are being used and try again to call **checkResourceNeed** after a certain time. If one or both resources are not in the available chopstick list, the consumer will try the checking job again after a certain length of time. When the consumer finishes using the resource, it calls the **put** method on both **Resource** unihandles, and then the **Resource** calls **sayAva** on the omnihandle to notify every consumer it is available to use now. So the **Resource** unihandle is back to available resource list on every consumer's device.

Deadlock is successfully prevented.

3.3.2.2 Design based on RMI mechanism

Pattern Name

RMI Shared Resource Allocation **(a documenting design pattern [7])**

Intent and Motivation

The ***RMI Shared Resource Allocation*** pattern applies to situations in which regular computing devices (PC, Laptop, etc), running in internet or intranet environment, need to compete for some shared resource. Every device should have equal opportunity to use the shared resource and avoid deadlock.

Applicability (or Prerequisites)

The ***RMI Shared Resource Allocation*** pattern is used in internet or intranet environment for fixed number devices competing for shared resource, which has a powerful central server, network administration and complicated routing protocol. It is suitable for big set of fixed users and resource, regular computing devices (PC, Laptop, etc) and large working space.

Solution

Every consumer and resource acts as a client part for this ***RMI Shared Resource Allocation*** system. Before a consumer or resource starts to work, its device gets an IP and

port from network administrator. Then it looks up the RMI server with the host IP and port. After the connection is established, every client should apply a userID from the RMI server then it can call the methods like local method invocation.
In the Server side, whenever it receives a resource-occupied request from a consumer, it looks up the resource list to check the availabilities of all the resources that the consumer needs. Only if both of them are available, the server will assign the resources to the consumer and block all the other requests to those resources. Then the server establishes a TCP connection to the resource and gets the consumer's functions request done. After the process is done, the server releases the resource so consumers can use the resource again. The server needs to provide an interface to declare all the functions it can provide.

Participants

- Consumer Client class, which applies a membership from RMI server, and has the member functions like resource-occupied request.
- Resource Client class, which applies a membership from RMI server, and provides the member functions of services.
- Server class, which provides available resource checking, resource allocation, and resource release, etc.
- Server interface class, which provides interface for all the member functions in server class.

3.3.3 RMI-based Design vs. M2MI-based Design

Comprising to RMI-based design of appointment making system, the M2MI-based design has the following differences:

- When a consumer needs resource, it needs to send messages to a central server, while M2MI broadcasts the request to all consumers and resources at the same time.
- RMI needs a powerful code-base Server, while M2MI is server less
- RMI needs to pre-compile code, while M2MI requires every device has the classes ahead of time
- RMI needs IP and port assignment and network administration, while M2MI is designed based on ad hoc environment and does not need that
- RMI needs exactly membership, while M2MI can allow devices come and go free without notification
- RMI simplify the development for client application, while M2MI simplifies the development of application for programmer
- RMI has 100% reliable data transfer, while M2MI don't have that.

4. Deliverables

All of the following principal deliverables will be contained in a CD:

- Source code
- Executive files
- Relevant documentation written in JavaDoc format
- Technical report, containing explanations of the system design and results from the tests conducted. A hard copy will be provided as well.
- User Manual, come with the Technical Repoter.

I will do a final demonstration of the tests in the Computer Science Lab.

5 Schedule

Oct 2002	Started to choose topic and write proposal
11/18/2002	Topic selected and registered for MS proposal seminar
12/20/2002	The rough version of my proposal was done
1/06/2003	The version 1.0 of my proposal was done
1/13/2003	The version 2.0 of my proposal was done
1/15/2003	The version 3.0 of my proposal was done
2/05/2003	The version 4.0 of my proposal was done
2/20/2003	The version 5.0 of my proposal was done
3/22/2003	The version 6.0 of my proposal was done
4/01/2003	The version 1.0 of Chat system template was done
4/07/2003	The version 2.0 of Chat system template was done
4/10/2003	The version 3.0 of Chat system template was done
4/15/2003	The final version of Chat System (4.0) template was approbated
4/17/2003	The version 7.0 of my proposal was done
4/24/2003	The final version of my proposal (8.0) was approbated
5/05/2003	Presentation for my proposal
May 2003	Code design
June 2003	Complete code design
July 2003	Detailed design and system test
August 2003	Technical report writing and preparing for the defense
Summer/2003	Defense

6 References

[1] Hans-Peter Bischof and Alan Kaminsky. “Many-to-Many Invocation: A new framework for building collaborative applications in ad hoc networks.”
CSCW 2002 Workshop on Ad Hoc Communication and Collaboration in Ubiquitous Computing Environments, New Orleans, Louisiana, USA, November 2002.

[2] Alan Kaminsky and Hans-Peter Bischof. “Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems.”
17th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2002), Onward! Track, Seattle, Washington, USA, November 2002

[3] Alan Kaminsky and Hans-Peter Bischof. “Many-to-Many Invocation: A new paradigm for ad hoc collaborative systems.”
Technical Report TR-2002-01, Rochester Institute of Technology, IT Lab, February 6, 2002.

[4] Alan Kaminsky. “Infrastructure for distributed applications in ad hoc networks of small mobile wireless devices.”
Technical Report, Rochester Institute of Technology, IT Lab, May 22, 2001.

[5] R. Stevens, Unix Network Programming, Volume 1, 2nd Edition, Prentice Hall PTR, 1998

[6] C.A.R Hoare, *Communicating Sequential Processes*, Oxford University, 1990

[7] http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/