GROUP COMMUNICATION (APPLICATION-LEVEL MULTICAST)

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu
Code available at: https://github.com/pdewan/ColabTeaching
PRE-REQUISITES

- Model-Interactor Separation
- Interaction Types
- Model Types
FROM 1-USER TO COLLABORATIVE

1-User App (Word)

Collaboration Functions

Collaborative Application (GoogleDocs)

Coupling

U1

U2

U3
ECHOER TO IM

Please enter an input line or quit or history
The woods are lovely dark and deep
The woods are lovely dark and deep
Please enter an input line or quit or history
But I have promises to keep
And miles to go before I sleep
history
The woods are lovely dark and deep, But I have promises to keep, And miles to go before I sleep

Please enter an input line or quit or history

Please enter an input line or quit or history
The woods are lovely dark and deep
[Alice]The woods are lovely dark and deep
Please enter an input line or quit or history
[Bob]But I have promises to keep
[Cathy]And miles to go before I sleep
history
[Alice]The woods are lovely dark and deep, [Bob]But I have promises to keep, [Cathy]And miles to go before I sleep

Please enter an input line or quit or history
public class ALauncherOfIM extends AConsoleModelBasedLauncher implements LauncherOfIM {
    public Class[] mainClasses() {
        return new Class[] {
            sessionManagerClass(),
            aliceClass(),
            bobClass(),
            cathyClass()
        };
    }
}

Can do executeAll or double click on each class in sequence

Can simply run each class from programming environment
ANATOMY / ARCHITECTURE

Architecture = Program components and their interaction

Components = objects?
Distributed Processing

Local Output Displaying Process

Local Input Intercepting Process

Local Output Displaying Process

Other processes

Direct or indirect distributed comm.
A process is created by the Operating System each time a program (in Java a main method) is run.

Same program may be executed multiple times to create multiple processes.
**PROCESS vs OBJECT vs DISTRIBUTED ARCHITECTURE**

Process architecture describes the processes that implement some potentially distributed application and the communication among these processes.

Object architecture describes the objects and the communication among objects in a process.

Distributed architecture maps processes to computers.

In our demos and testing we will map them all to one machine.
Single-User ➔ Collaborative Architecture

Replace local calls with “transparent” remote calls?

Remote calls are not transparent – must at least deal with communication errors

Blocking call and round trip delay to get local feedback

Central bottleneck which may not always be available

No awareness of others

Sharing at the model level

Architecture is too constrained!

Put the model on one machine and an interactor of a user on his/her machine
Nothing said about other processes and the nature of communication
Commonality not exploited
ABSTRACTION LAYERS IN COLLABORATION TOOLKIT?
Inter-process communication allows two processes on different hosts to communicate with each other.

General form: send (host, port, message)
Process vs Object vs Distributed Architecture (Review)

Process architecture describes the processes that implement some potentially distributed application and the communication among these processes.

Object architecture describes the objects and the communication among objects in a process.

Distributed architecture maps processes to computers.

In our demos and testing we will map them all to one machine.
Interprocess Communication (Sockets, RMI, ..)

Inter-process communication allows two processes on different hosts to communicate with each other.

General form: send (host, port, message)
How did H² know about H¹ and H³?

Static (hardwired) dynamic “sessions”?
Dynamic Sessions

send(h^1, p^1, \{h^2, p^2\})

send(h^1, p^1, \{u^3, p^3\})

send(h^2, p^2, \{h^1, p^1\})

send(h^2, p^2, \{h^3, p^3\})

send(h^3, p^3, \{h^1, p^1\}, \{h^2, p^2\})

send(h^{SM}, p^{SM}, h^1, p^1)

send(h^{SM}, p^{SM}, h^2, p^2)

send(h^{SM}, p^{SM}, h^3, p^3)
**Dynamic P2P**

IM Session Manager

**IPC**  
```
send(h^1, p^1, msg)
```

**IPC**  
```
send(h^3, p^3, msg)
```

**IM**

```
U^1
```

```
U^2
```

```
U^3
```
Many reasons for using relayed sessions (consistency, security, performance)

Higher-level abstraction than IPC?
ABSTRACTION LAYERS IN COLLABORATION TOOLKIT?

- Interprocess Communication (Sockets, RMI, ..)
- Group Communication (Multicast)
- ???
App-Specific Session Manager

1. \(\text{send}(h^1, p^1, \{h^2, p^2\})\)
2. \(\text{send}(h^1, p^1, \{h^3, p^3\})\)
3. \(\text{send}(h^{SM}, p^{SM}, h^1, p^1)\)

4. \(\text{send}(h^2, p^2, \{h^1, p^1\})\)
5. \(\text{send}(h^2, p^2, \{h^3, p^3\})\)
6. \(\text{send}(h^{SM}, p^{SM}, h^2, p^2)\)

7. \(\text{send}(h^3, p^3, \{h^1, p^1\}, \{h^2, p^2\})\)
8. \(\text{send}(h^{SM}, p^{SM}, h^3, p^3)\)

\(\text{IM Session Manager}\)
**App-Specific Session Manager (No Callbacks)**

- **U¹**
  - `send(h^{SM}, p^{SM}, h^1, p^1)`
- **U²**
  - `send(h^{SM}, p^{SM}, h^2, p^2)`
- **U³**
  - `send(h^{SM}, p^{SM}, h^3, p^3)`
**Generic Session Manager (One per Collaborative “Session”)**

Should clients have to know about ports or send its host?
**Generic Session Manager (Client Library, One per Collaborative Session)**

Generic Session Manager will have to be told about session manager host for this application.

- **Library**
  - IPC send replaced by group comm. join
  - Group Comm.
  - Local library at each site
  - Group Comm.

- **Group Comm.**
  - As before session manager can send host and port number to application
  - send(h³, p³, {h¹, p¹}, {h², p²})
  - Group Comm.

- **U¹**
  - join(h²SM)

- **U²**
  - join(h²SM)

- **U³**
  - join(h²SM)

- **IM**

Library could fill in port a la Web browser and can also register own port and send its host.
**Dynamic P2P, No Multicast**

Better communication support?

Generic Session Manager

Group Comm.

send($h^1, p^1, msg$)

IM

$U^1$

Group Comm.

send($h^3, p^3, msg$)

IM

$U^2$

Group Comm.

IM

$U^3$
Dynamic P2P Application-Level Multicast

Application-level multicast: Multiple messages delivered to network layer

Generic Session Manager

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Group Comm. to Others(msg)

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer

Application - level multicast: Multiple messages delivered to network layer

Single message delivered to network layer which can result in a single message being put on the wire for multiple destinations

Generic Session Manager

Application code does not have to define a session participant callback and maintain information about participants

Application code can be unaware of specific users

Multiple messages delivered to network layer
**Dynamic Sessions (Relayed)**

Generic Session Manager

Group Comm.

- IM
  - msg
  - $U^1$

Group Comm.

- IM
  - msg
  - $U^2$

Group Comm.

- IM
  - msg
  - $U^3$

`toOthers(msg)`
**Flexible Group Communication**

Same logical call made in both relayed and p2p case

Generic Session Manager

Group Comm.  →  toOthers(msg)  →  Group Comm.

IM  →  msg  →  U¹

IM  →  msg  →  U²

IM  →  msg  →  U³

Library can support both forms of communication

It can choose based on security and performance characteristics (but not consistency)

It can let application choose
**SPECIFYING ROUTE AT JOIN TIME: P2P**

- **Generic Session Manager**
  - **Group Comm.**
    - **IM**
      - **U^1**
    - **toOthers(msg)**
      - **msg**
    - **join(h^{SM}, Direct)**
  - **Group Comm.**
    - **IM**
      - **U^2**
    - **msg**
  - **Group Comm.**
    - **IM**
      - **U^3**
    - **msg**

- **Sent Directly**
  - Each site chooses independently
SPECIFYING ROUTE AT JOIN TIME: RELAYED

Generic Session Manager

Group Comm.

Group Comm.

Group Comm.

toOthers(msg)

IM

IM

IM

msg

msg

msg

U¹

U²

U³

join(h^{SM}, Relayed)

Relayed communication
**ADDITIONAL MULTICAST GROUPS?**

- All session members other than the caller

**Diagram:***
- **Generic Session Manager**
  - **Group Comm.**
    - **Group Comm.**
      - **Group Comm.**
        - **IM**
          - **U₁**
        - **IM**
          - **U²**
        - **IM**
          - **U³**
    - **toOthers(msg)**

- Connections shown with arrows indicating flow of messages (msg) from one session member to others.
TO\textsc{ALL}

All session members including the caller

\textbf{Generic Session Manager}

\textbf{Group Comm.}

\textbf{IM}

\textbf{msg}

\textbf{U}^1

\textbf{toAll(msg)}

\textbf{Group Comm.}

\textbf{IM}

\textbf{msg}

\textbf{U}^2

\textbf{Group Comm.}

\textbf{IM}

\textbf{msg}

\textbf{U}^3
MUD (Multi-User Dungeons): Other Groups

- **say** | "your utterance here"
  Everyone in the room can 'hear' what you 'say', or see what you type.

- **whisper** *playername* = *your whisper here*
  ... so only the player(s) named, and in the room, can hear your whisper.

- **mutter** *player* = *message*
  Mutters *message* to everyone in the same room EXCEPT *player*.

- **join**(*hSM*, *u*, *Relayed*)
  Logical user id (credentials) rather than physical host given as argument

- **toOthers**(*msg*)
  User-aware application code but not host and port aware

Single Session per Session Manager

Multiple sessions?

Generic Session Manager

Group Comm.
- IM
  - msg
  - U¹

Group Comm.
- IM
  - msg
  - U²

- toOthers(msg)

Group Comm.
- IM
  - msg
  - U³

join(h^{SM}, u^2, Relayed)
Single Session Per Session Manager
**MULTIPLE SESSIONS PER SESSION MANAGER**

- Session members of one IM should not know or be notified about members of other IMs.
- In relayed, messages not relayed to members of other session.
- Join now requires access control, a user may be allowed a subset of sessions.

**Generic Session Manager**

- Join now requires access control, a user may be allowed a subset of sessions.
- Fewer servers.
- Less setup overhead.
- Shared session directory.

**Join**

- \( \text{join}(h^\text{SM}, s^2, u^2, \text{Relayed}) \)
- \( \text{join}(h^\text{SM}, s^1, u^2, \text{Relayed}) \)
ASymmetric vs. SYMMETRIC JOINS

Create call?

Asymmetric or multiple programs: One user creates and joins and other users join and race conditions

Single symmetric program: If session does not exist, create it, and join; otherwise simply join

 join\left(h^{SM}, s^2, u^2, \text{Relayed}\right)

join\left(h^{SM}, s^1, u^2, \text{Relayed}\right)
MULTIPLE SESSIONS?
**MULTIPLE SESSIONS?**

Are text chat, code share and audio video separate sessions?

They are different application sub sessions in same session, no access control required to join applications once user is in session, and each user knows about users of each application.
Need to keep users of applications separate so that multicast calls can distribute messages correctly, so users in application sub-sessions. Users in session can interact with any application in session – access control done when user joins session. User notified of all applications added to session and all users interacting with the application.
MULTIPLE APPLICATIONS PER SESSIONS

join(\(h^{SM}, s^1, u^2\), Editor, Relayed)

If session not created, create it

If user not in session, add it

If application != null and not in session, add it

If application != null add user to application sub-session

join(\(h^{SM}, s^1, u^2\), IM, Relayed)
FLAT SESSIONS (REVIEW)

Session

Alice

Bob

Cathy

join(h^{SM}, s^1, Cathy, Relayed)

Some process running on behalf of Cathy joins the session

Session manager oblivious of application implemented by process

Session manager oblivious of application implemented by process
Motivating Sub-sessions

Similar sub-sessions supported by Google Hangout, LiveMeeting, WebEx, ....
**Session with Application Sub-Sessions**

Session

- **Alice**
- **IM**
- **Bob**
- **Editor**
- **Cathy**
- **David**

**Join session without joining any subsession?**

David’s process cannot send or receive messages and simply listens to session callbacks, which can inform its user of session activity and join application sub-sessions.

- `join(h^{SM}, s^1, Cathy, IM, Relayed)`
- `join(h^{SM}, s^1, Cathy, Editor, Relayed)`
- `join(h^{SM}, s^1, David, null, Relayed)`
SESSION JOIN SEMANTICS

join(h^{SM}, s^1, u^2, Editor, Relayed)

- If session not created, create it
- If user not in session, add it
- If application not in session, add it
- Add user to application sub-session

join(h^{SM}, s^1, u^2, IM, Relayed)
SYNCHRONIZATION

Generic Session Manager

Group Comm.

Group Comm.

Group Comm.

toOthers(msg)

IM

IM

IM

msg

msg

msg

joined (s^1, u^2, IM, newSession?, newApp?)

received (u^2, msg)

join(h^{SM}, s^1, u^2, IM, Relayed)

When do these calls/callbacks return: Synchronous vs. asynchronous?

Semantics specified?
**Synchronous vs. Asynchronous**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operation(&lt;params&gt;)</code></td>
<td>Synchronous: Operation invoker waits until the operation “finishes”</td>
</tr>
<tr>
<td><code>write(file, data)</code></td>
<td>Asynchronous: Operation invoker does not wait until completion</td>
</tr>
<tr>
<td><code>toOthers(msg)</code></td>
<td>Some other operation (e.g. callback) needed to wait for result or completion status</td>
</tr>
</tbody>
</table>
Synchronous vs. Asynchronous vs. Blocking Operations

- **operation(<params>)**
- **write(file, data)**
- **toOthers(msg)**

**Blocking:** Operation invoker waits, unblocking possibly before, until, or after operation completion (e.g. when data given to local OS)

- Synchronous is always blocking
- Blocking is not always synchronous
**Operation Completion: Calls vs. Callbacks**

- **Generic Session Manager**
  - joined (s₁, u², IM, newSession?, newApp?)
  - received (u², msg)

- **Group Comm.**
  - IM
    - toOthers(msg)
  - IM
    - msg
  - IM
    - msg

- **Group Comm.**
  - IM
    - join(h^{SM}, s₁, u², IM, Relayed)

- **Callback finishes when it finishes execution (at the other site)**
- **Call finishes when the call and any associated callbacks finish**
CALLS AND CALLBACKS: SYNC VS ASYNC?

joined (s₁, u², IM, newSession?, newApp?)

Generic Session Manager

received (u², msg)

Group Comm.

toOthers(msg)

IM

msg

U¹

Synchronous call: Local response effected

Synchronous call/callback: Local and remote response affected

join(h^{SM}, s₁, u², IM, Relayed)

IM

msg

U²

IM

msg

U³
Inter-Layer Dependencies

If lower-level layer, is asynchronous can we make higher-level layer synchronous?

Yes, send explicit acks (TCP ~ UDP)

Group Communication (Multicast)

If lower-level layer, is synchronous can we make higher-level layer asynchronous?

“Yes”, with separate threads
**MULTIPLE THREADS**

- Requesting Thread 1
- Sending Thread
- Producers
- Bounded Buffer
- Consumers

- Requesting thread passes message to sending thread and does not block
- Sending thread invokes synchronous operation
- Communication between the two threads example of some classical problem?
- How many consumers?
- Consistency constraints?

What if the two threads work at different rates (expected) – what if more than one pending message?
# Multiple Threads

- Requesting Thread 1
- Sending Thread
- Producers
- Bounded Buffer
- Consumers

### Notes:
- # producers is application specific and cannot be controlled
- > 1 consumers can result in message reordering
Only messages within an application sub-session need to be ordered
**How Many Consumers**

- **Requesting Thread 1**
- **Sending Thread**

- **msg¹**
- **msg²**
- **msg³**
- **msg⁴**

**Asynchronous**

**Bounded Buffer Per Application Sub-session**

**Synchronous**

- One sending thread per application sub session at both session server and its clients
- Multicast layer hides this from client
**Feedback vs Feedthrough**

- **Requesting Thread 1**
  - msg¹
  - msg²
  - msg³
  - msg⁴

- **Sending Thread**

**Asynchronous**

- Bounded Buffer Per Application Sub-session

**Synchronous**

- Requesting thread does not block, so feedback does involve round trip delay
- Messages to server and other clients separated by round trip delays
SYNCHRONOUS IPC

How bad is this in practice?

Generic Session Manager

Group Comm.
IM

U1
msg1
ack1

U2
msg2
ack2

U3
msg2
ack2

Round trip time ~ 100-2000ms

Time between telepointer movements: 10 ms

Noticeable (tolerable) latency: 30ms, 500ms
MESSAGE RECEIPTS?

Threads that receive messages?

Group communication layer handles these threads
How many Receiving Threads?

Server can have one thread for all messages, which orders the messages and then simply forwards them into the appropriate sending threads, which does real work.

Messages received for an app subsession should be handled by one thread.

Client can have one thread per application subsession.

How many client and server threads?
THREADS (CONCRETE EXAMPLE)

- Received Message Consumer in Server
- IM Application-session sender
- IM Application-session sender
- IM Application-session Receiver
**Sending Threads: BOB Client (Relayed)**

Application-session threads send join and leave requests to session manager and also relay messages to session manager, waiting if necessary based on delay parameters.
Sending Threads: Bob Client (P2P)

Application-session sending threads send join and leave requests to session manager and also serialize messages of the session, forwarding them to peer user threads.

Peer sending threads receive messages from their application-session threads and send messages to peers, delaying messages based on delay parameters.

Could share peer threads among application-session threads but more modular to create new threads.
**Sending Threads: Session Server**

My implementation does not have per user thread at server.

If feedthrough is an issue, use direct communication.

A server may have numerous sessions, so per user thread maybe too much overhead.

Moral: in production version do not use blocking IPC such as RMI.
Concrete Threads (New Version)

- Received Message Consumer in Server
- IM Application-session sender
- IM Application-session Receiver
- IM Application-session Sender
- IM Application-session, Peer Senders
- No peer senders for Bob as it is using relayed communication
TOALL (RELAYED)

Generic Session Manager

Group Comm.

Group Comm.

Group Comm.

IM

msg

U1

IM

msg

U2

IM

msg

U3

Do we need toAll() if messages are not relayed?
**TOALL(P2)**

Both local feedback and remote feedback in callbacks

Both local feedback and remote feedback in callbacks

Generic Session Manager

Group Comm.

Group Comm.

Group Comm.

IM

U^1

msg

U^2

msg

U^3

msg

toAll(msg)
The Concept of Group Communication

- Interprocess Communication (Sockets, RMI, ..)
- Group Communication (Multicast)

???
GROUP MESSAGES DESIGN

Interprocess Communication (Sockets, RMI, ..)

GroupMessages

Group Messages Implementation

Easier to code as it is RPC, but synchronous, but feedthrough is an issue

Motivation for asynchronous RPC
Example (UI)
@Tags({DistributedTags.SERVER, DistributedTags.SESSION_MANAGER, ApplicationTags.IM})
public class SessionManagerServerStarter {
    static ASessionManager server;
    public static void main (String[] args) {
        //do tracing
        ...
        server = new ASessionManager();
        server.register(); //with RMI server
    }
}

Annotations provide typed comments like traces but are passive
@Tags({DistributedTags.CLIENT_1, ApplicationTags.IM})
public class AliceIM implements ExampleIMSession{
public static final String USER_NAME = DistributedTags.CLIENT_1;
public static void main (String[] args) {
    String[] launcherArgs = {
        SESSION_SERVER_HOST, SESSION_NAME,
        USER_NAME, APPLICATION_NAME, Communicator.DIRECT
    };
    //do tracing
    ...
    (new AnIMClientComposerAndLauncher()).
    composeAndLaunch(launcherArgs);

    join(hSM, s₁, u², Editor, Direct)

    Shared symmetric program

    Host specification?

    IM
Computer name, domain, and workgroup settings

- Computer name: dewantab
- Full computer name: dewantab.cs.unc.edu
- Computer description: LV-C228X dewantab
- Domain: cs.unc.edu

```java
public interface ExampleIMSession {
    public static final String SESSION_NAME = "FrostySession";
    public static final String APPLICATION_NAME = "IM";
    public static final String SESSION_SERVER_HOST = "localhost";
}
```

localHost allows you to test same program on different hosts
Error Cause?

java.rmi.ConnectIOException: Exception creating connection to: 172.17.1.182; nested exception is:
    java.net.SocketException: Permission denied: connect
    at sun.rmi.transport.tcp.TCPEndpoint.newSocket(TCPEndpoint.java:631)
    at sun.rmi.transport.tcp.TCPChannel.createConnection(TCPChannel.java:216)
    at sun.rmi.transport.tcp.TCPChannel.newConnection(TCPChannel.java:202)
    at sun.rmi.server.UnicastRef.invoke(UnicastRef.java:129)
    at java.rmi.server.RemoteObjectInvocationHandler.invoke(RemoteObjectInvocationHandler.java:148)
    at com.sun.proxy.$Proxy0.newMessage(Unknown Source)
    at util.session.AMessageSenderRunnable.run(AMessageSenderRunnable.java:71)
    at java.lang.Thread.run(Thread.java:744)
The local host has a different meaning when you are using Cisco VPN!

Will get “access error” if Cisco VPN is connected

Giving host name directly does not help

Do not use Cisco VPM if using Cisco VPN!
@Tags({DistributedTags.CLIENT_1, ApplicationTags.IM})
public class AliceIM implements ExampleIMSession{
public static final String USER_NAME = DistributedTags.CLIENT_1;
public static void main (String[] args) {
    String[] launcherArgs = {
        SESSION_SERVER_HOST, SESSION_NAME, USER_NAME, APPLICATION_NAME, Communicator.DIRECT};
//do tracing
...
    (new AnIMClientComposerAndLauncher()).composeAndLaunch(launcherArgs);
}
public void compose(String[] args) {
    communicator = createCommunicator(args);
    super.compose(args);
    addCollaborationFunctions();
    doJoin();
}

protected void doJoin() {
    communicator.join();
}

Instantiate local group communication library
Add callbacks, before connecting to server
Invoke call on server

Should make createCommunicator as part of library
**Multicast Arbitrary Serializable Objects**

```java
public synchronized void replicatedAdd(ElementType anElement) {
    int anIndex = size();
    super.observableAdd(anIndex, anElement);
    if (communicator == null) return;
    ListEdit listEdit = new AListEdit<
        ElementType>(OperationName.ADD,
        anIndex, anElement, ApplicationTags.IM);
    communicator.toOthers(listEdit);
}
```

```java
public interface ListEdit<ElementType> extends Serializable {
    int getIndex();
    void setIndex(int anIndex);
    ElementType getElement();
    void setElement(ElementType anElement);
    ...
}
```

RMI uses Add locking to project layer which requires the communicated objects to be labelled as Serializable, variables of only serializable superclasses can be communicated remotely.
Example (UI) (Review)
@Tags({DistributedTags.SERVER, DistributedTags.SESSION_MANAGER, ApplicationTags.IM})

public class SessionManagerServerStarter {
    static ASessionManager server;
    public static void main (String[] args) {
        //do tracing

        server = new ASessionManager();
        server.register(); //with RMI server
    }
}
@Tags({DistributedTags.CLIENT_1, ApplicationTags.IM})
public class AliceIM implements ExampleIMSession{
public static final String USER_NAME = DistributedTags.CLIENT_1;
public static void main (String[] args) {
    String[] launcherArgs = {
        SESSION_SERVER_HOST, SESSION_NAME, 
        USER_NAME, APPLICATION_NAME, Communicator.DIRECT};
    //do tracing
    ...
    (new AnIMClientComposerAndLauncher()).composeAndLaunch(launcherArgs);
JOIN CALL IN SHARED PROGRAM

```java
public void compose(String[] args) {
    communicator = createCommunicator(args);
    super.compose(args);
    addCollaborationFunctions();
    doJoin();
}

protected void doJoin() {
    communicator.join();
}
```

- Instantiate local group communication library
- Add callbacks, before connecting to server
- Invoke call on server
- Should make createCommunicator as part of library
MULTICAST ARBITRARY SERIALIZABLE OBJECTS

```java
public synchronized void replicatedAdd(ElementType anElement) {
    int anIndex = size();
    super.observableAdd(anIndex, anElement);
    if (communicator == null) return;
    ListEdit listEdit = new AListEdit<
        ElementType>(OperationName.ADD,
        anIndex, anElement, ApplicationTags.IM);
    communicator.toOthers(listEdit);
}
```

RMI uses Add locking to project layer which requires the communicated objects to be labelled as Serializable, variables of only serializable superclasses can be communicated remotely.

Add operation marshalled into serializable object, reverse process at receiver

Programmer does marshalling and unmarshalling as multicast RPC does not exist and is hard to implement
protected void addHistoryInCoupler() {
    historyInCoupler = new AHistoryInCoupler(history);
    communicator.addPeerMessageListener(historyInCoupler);
}

public class AHistoryInCoupler implements PeerMessageListener {
    protected SimpleList<String> history;
    public AHistoryInCoupler(SimpleList<String> theEchoer) {
        history = theEchoer;
    }
    public void objectReceived(Object message, String userName) {
        if (message instanceof ListEdit) {
            processReceivedListEdit((ListEdit<String>) message, userName);
        }
    }
}

Can have multiple receive listeners processing different kinds of messages

Unmarshalling
Calls vs Callbacks

joined (s₁, u², IM, newSession?, newApp?)

toOthers(msg)

Please enter an input line or quit or history
The woods are lovely dark and deep
[Alice]The woods are lovely dark and deep
Please enter an input line or quit or history
[Bob]But I have promises to keep
[Cathy]And miles to go before I sleep
history

join(h^{SM}, s₁, u², IM, Relayed)

Join callback not needed in this application
UNAWARE SYMMETRIC JOIN

Somehow all users know when to join

Could join when we know someone else has created and joined session

No awareness of out of band activities
**Session with Application Sub-Sessions**

Join session without joining any subsession?

David’s process cannot send or receive messages and simply listens to session callbacks, which can inform its user of session activity and join application sub-sessions.
### User Awareness

<table>
<thead>
<tr>
<th>Common</th>
<th>AMainClassListLauncher</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>SessionManagerServerStarter</td>
</tr>
<tr>
<td>2:</td>
<td>AliceIMJoiner</td>
</tr>
<tr>
<td>3:</td>
<td>BobIM</td>
</tr>
<tr>
<td>4:</td>
<td>CathyIM</td>
</tr>
</tbody>
</table>

Alice now joins session when after it has been joined by some one else (invitation based joining) and is informed about session activities.
SERVER AND ALICEIMJOINER

Session aware Joiner UI

No prompt, this is not the IM user interface
SESSION WITH APPLICATION SUB-SESSIONS

Session

Alice
SESSION WITH APPLICATION SUB-SESSIONS

Session

Alice → IM → Bob

Alice → Bob
SERVER AND ALICEIMJOINER

Session aware Joiner UI
Bob Joins

Session aware Joiner UI

IM UI
SESSION MANAGER TRACE

Session

Alice  IM  Bob

Alice  Bob
public class AliceIMJoiner extends AliceIM{
public static void main (String[] args) {
    String[] launcherArgs = {SESSION_SERVER_HOST,
         SESSION_NAME, USER_NAME, null, Communicator.DIRECT};
    (new AJoiningIMComposerAndLauncher()).compose(launcherArgs);
}
}

public void addAwareness() {
    sessionJoiner = new ADynamicSessionJoiner
    (sessionServerHost, userName);
    communicator.addSessionMessageListener(sessionJoiner);
}

A different program is run by Alice’s session aware joiner
public class ADynamicSessionJoiner implements SessionMessageListener {

    ...

    public void clientJoined(String aUserName, String anApplicationName, String aSessionName, boolean aNewSession, boolean aNewApplication, Collection<String> anAllUsers) {
        printAwarenessMessage(aUserName, anApplicationName, aSessionName, aNewSession, aNewApplication, anAllUsers);

        if (aNewApplication && anApplicationName != null && DEFAULT_APPLICATION_NAME.equals(anApplicationName))
            joinSession(anApplicationName, aSessionName);
    }

    }
SESSION MANAGER/COMMUNICATOR STEPS

- ClientJoinFinished.java
- ClientJoinInformationUpdated.java
- ClientJoinInitiated.java
- ClientJoinNotificationDistributedToListeners.java
- ClientJoinNotificationMarshalled.java
- ClientJoinNotificationReceived.java
- ClientJoinNotificationUnmarshalled.java
- ClientLeaveInformationUpdated.java
- ClientLeaveNotificationDistributedToListeners.java
- ClientLeaveNotificationMarshalled.java
- ClientLeaveNotificationReceived.java
- ClientLeaveNotificationUnmarshalled.java
- ClientReceivedObjectUnmarshalled.java
- DataReceiveMarshalled.java
- DelayedMessageInfo.java
- DelayVariationSet.java
- JoinRequest.java
- JoinRequestMarshalled.java
- LeaveRequest.java
- LeaveRequestMarshalled.java
- MessageBuffered.java
- MessageBufferInfo.java
- MessageBufferReferenceCountDecrementedException.java
- MessageCopied.java
- MessageForwarded.java
- MessageGivenToFilter.java
- MessageInfo.java
- MessagePutInQueue.java
- MessageReceived.java
- MessageRetrievedFromQueue.java
- MessageSent.java
- MessageUnBuffered.java
- MinimumDelaySet.java
- MulticastGroupCreated.java
- MulticastGroupJoinInformationUpdated.java
- MulticastGroupLeaveInformationUpdated.java

Not all important steps traced and at least one step deprecated
SESSION MANAGER/COMMUNICATOR STEPS (CONTD)
SUMMARY

- Distributed architecture = process + object architecture
- General and special distributed architectures exists, which depend on distributed communication layer
- At this point, looking at general architecture
- IPC provides the most general architecture but maybe too general, not providing support for
- With IPC need to build
  - own session manager for dynamic sessions, group multicast, choice between relayed and direct communication, threading
- Can build group communication automating this
- Session vs. application-session
- Relayed vs direct communication
- All, specific user, all multicast groups
- Synchronous vs. asynchronous
- Threads to balance latency vs. consistency
- Symmetric vs asymmetric join
- Serialization and marshalling
- Session awareness
NEXT

- How to use group communication for different classes of applications
  - Model-based sharing
  - Window-based sharing
- How do build higher level abstractions for these classes?