Work groups meeting – 5

INF5040 (Open Distributed Systems)

Sabita Maharjan
sabita@simula.no

Department of Informatics
University of Oslo

October 19, 2009

Outline

- Spread
- Spread C API
- Spread Java API

Group Communication System

- Services provided by group communication systems:
  - Abstraction of a Group
  - Multicast of messages to a Group
  - Membership of a Group
  - Reliable messages to a Group
  - Ordering of messages sent to a Group
  - Failure detection of members of the Group
  - A strong semantic model of how messages are handled when changes to the Group membership occur

Spread

- An open source toolkit that provides a high performance messaging service that is resilient to faults across local and wide area networks
- Does not support very large groups, but does provide a strong model of reliability and service such as ordering
- Integrates a membership notification service into the stream of messages
- Supports multiple link protocols and multiple client interfaces
- The client interfaces provided with Spread include native interfaces for Java and C

Spread

- Provides different types of messaging services to applications
  - messages to entire groups of recipients
  - membership information about who is currently alive and reachable
- Provides both ordering and reliability guarantees

Level of Service

- When an application sends a Spread message, it chooses a **level of service** for that message.
- The level of service selected controls what kind of ordering and reliability are provided to that message.

Spread Service Type	Ordering	Reliability
UNRELIABLE_MESS	None	Unreliable
RELIABLE_MESS	None	Reliable
FIFO_MESS	FIFO by Sender	Reliable
CAUSAL_MESS	Causal (Lamport)	Reliable
AGREED_MESS	Total Order (Consistent w/Causal)	Reliable
SAFE_MESS	Total Order	Safe

Ordering

- None
  - No ordering guarantee
  - Any other message sent with “None” ordering can arrive before or after this message
- FIFO by sender
  - All messages sent by this connection of FIFO ordering are delivered in FIFO order
- Causal (Lamport)
  - All messages sent by all connections are delivered in an order consistent with “Causal” order (Lamport)
  - Consistent with FIFO ordering
- Total Order (Consistent w/Causal)
  - All messages sent by all connections are delivered in the exact same order to all recipients
  - Consistent with Causal order

Reliability

- Unreliable
  - The message may be dropped or lost
  - The message will not be recovered by Spread
- Reliable
  - The message will be reliably delivered to all recipients who are members of the group to which the message was sent
  - Spread will recover message to overcome any network losses
- Safe
  - The message will only be delivered to a recipient if the daemon that recipient is connected to knows that all Spread daemons have the message
  - If a membership change occurs, and as a result the daemon cannot determine whether all daemons in the old membership have the message, then the daemon will deliver the Safe message after a TRANSITIONAL_MEMBERSHIP message.

Spread C API

Spread Basics

- To access spread package from your application
  - #include <sp.h>

Connecting/Disconnecting

- To establish a connection to a spread daemon

```
int SP_connect( const char * spread_name, const char * private_name,
int priority, int group_membership, mailbox * mbox, char * private_group
);
```

0/1 flag
1➔ priority connection

Boolean integer: 0/1
1 ➔ The application receives group membership messages for this connection

Name of the spread daemon to connect to

A pointer that holds the mbox for the connection

Name of the connection

A pointer that contains the private group name for the connection

Connecting/Disconnecting

- **Return Values**
  - ACCEPT_SESSION: on success
  - ILLEGAL SPREAD: spread name given to connect to was illegal for some reason
  - COULD_NOT_CONNECT: lower level socket calls failed to allow a connection to the specified spread daemon right now
  - CONNECTION_CLOSED: during communication to establish the connection errors occured and the setup could not be completed
  - REJECT_VERSION: the daemon or library has a version mismatch
  - REJECT_NO_NAME: no user private name was provided
  - REJECT_ILLEGAL_NAME: name provided violated some requirement
  - REJECT_NOT_UNIQUE: name provided is not unique on this daemon

Connecting/Disconnecting

- To terminate the connection to the daemon

```
int SP_disconnect( mailbox mbox );
```

- **Return Values**
  - NORMAL: returns 0 on success
  - ILLEGAL_SESSION: when the session mbox given is not a valid connection

Joining/Leaving

- To join a group on the connection

```
int SP_join( mailbox mbox , const char * group );
```

- **Return Values**
  - NORMAL: returns 0 on success
  - ILLEGAL_GROUP: The group given to join was illegal
  - ILLEGAL_SESSION: the session specified by mbox is illegal
  - CONNECTION_CLOSED: during communication errors occured and the ”join” could not be initiated

Joining/Leaving

- To leave a group

```
int SP_leave( mailbox mbox, const char * group );
```

If the group does not exist among the Spread daemons this operation is ignored, otherwise the group is left.

- **Return Values**
  - NORMAL: returns 0 on success
  - ILLEGAL_GROUP: the group given to leave was illegal
  - ILLEGAL_SESSION: the session specified by mbox is illegal
  - CONNECTION_CLOSED: during communication errors occured and the ”leave” could not be initiated

Multicast and Family

- To multicast a message to one or more groups

- int SP_multicast(mailbox *mbox, service service_type, const char * group,* int16 *mess_type, int mess_len, const char * mess);*

- int SP_multigroup_multicast(mailbox *mbox , service service_type,* int *num_groups , const char groups[][MAX GROUP NAME],* int16 *mess_type, int mess_len, const char * mess);*

The message can be sent to only one group

The message can be sent to multiple groups

Multicasting

- **Return Values**
  - NORMAL: the number of bytes sent on success
  - ILLEGAL_GROUP: the mbox given to multicast on was illegal
  - ILLEGAL_SESSION: the message had an illegal structure
  - CONNECTION_CLOSED: during communication to send the message errors occured and the ”send” could not be completed

Receiving

- To receive a message

```
int SP_receive( mailbox mbox, service * service type, char sender[MAX GROUP_NAME], int max_groups, int * num_groups, char groups[][MAX GROUP_NAME], int16 * mess_type, int * endian_mismatch, int max_mess_len, char * mess );
```

receives both data messages and membership messages for the connection

which connection to receive a message on

message type of the message just received

The rest of the parameters differ in meaning depending on the *service type*

Receiving

- **Return Values**
  - NORMAL: returns the size of the message in success
  - ILLEGAL_SESSION: the mbox given to receive on was illegal
  - ILLEGAL_MESSAGE: the mbox had an illegal structure
  - CONNECTION_CLOSED: during communication to receive the message communication errors occured and the ”receive” could not be completed
  - BUFFER_ TOO_SHORT: the message body buffer was too short to hold the message being received
  - GROUPS_TOO_SHORT: the groups buffer was too short to hold the groups list or member list being received

Spread Java API

Java Interface to Spread Toolkit

- The Spread library consists of a package, "spread“
  - 10 classes.
- Main classes:
  - SpreadConnection, which represents a connection to a deamon,
  - SpreadGroup which represents a spread group
  - SpreadMessage, which represents a message that is either being sent or being received with spread.

Spread Basics

- The Spread package is contained in spread.jar
- To use Spread from a Java application, this file should be in your classpath.
  - CLASSPATH enviornment variable (the directory containing spread.jar should be in)
- For applets
  - put spread.jar in the same directory as the applet class.
- To access the Spread classes from any classes you write
  - Include **import spread.*;** at the top of the .java file.

Connecting/Disconnecting

- To establish a connection to a spread daemon

```
SpreadConnection connection = new SpreadConnection();
connection.connect(InetAddress.getByName("daemon.address.com"),
0, "privatename", false, false);
```

a class in the package java.net

Port to connect to

name or IP

- To terminate the connection to the daemon,

```
connection.disconnect();
```

Joining/Leaving

- To join a group on the connection

```
SpreadGroup group = new SpreadGroup();
group.join(connection, "group");
```

- Spread connection on which the group is joined
- Spread knows which connection messages should be received on

name of the group to join

- To leave a group

```
group.leave();
```

Multicasting

- To multicast a message to one or more groups

```
SpreadMessage message = new SpreadMessage();
```

This creates a new outgoing message

```
message.setData(data);
message.addGroup("group");
message.setReliable();
```

Message data

The groups the message is going to

Type of delivery requested

- To send the message

```
connection.multicast(message);
```

Receiving

- To receive a message

```
SpreadMessage message = connection.receive();
```

receive() will block until a message is available

```
if(message.isRegular())
   System.out.println("New message from " + message.getSender());
else
   System.out.println("New membership message from " +
message.getMembershipInfo().getGroup());
```

return a MembershipInfo object, which provides information about the membership change

Message Factory

- A utility included with the java interface to spread
- An object of the MessageFactory class is used to generate any number of outgoing messages based on a default message.

```
messageFactory = new MessageFactory(message);
```

- To change the default at a later time

```
messageFactory.setDefault(message);
```

- To get a message from the message factory

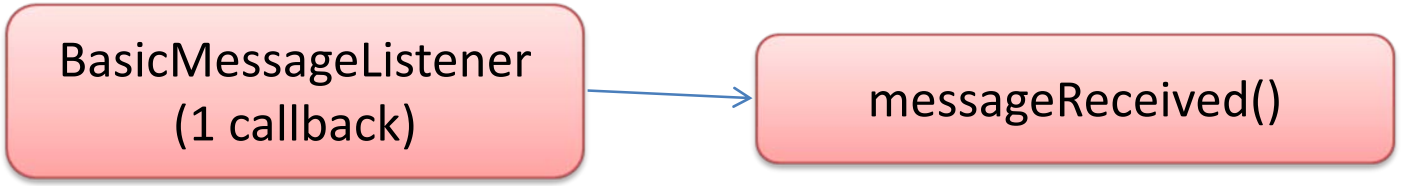
```
SpreadMessage message = messageFactory.createMessage();
```

Listeners

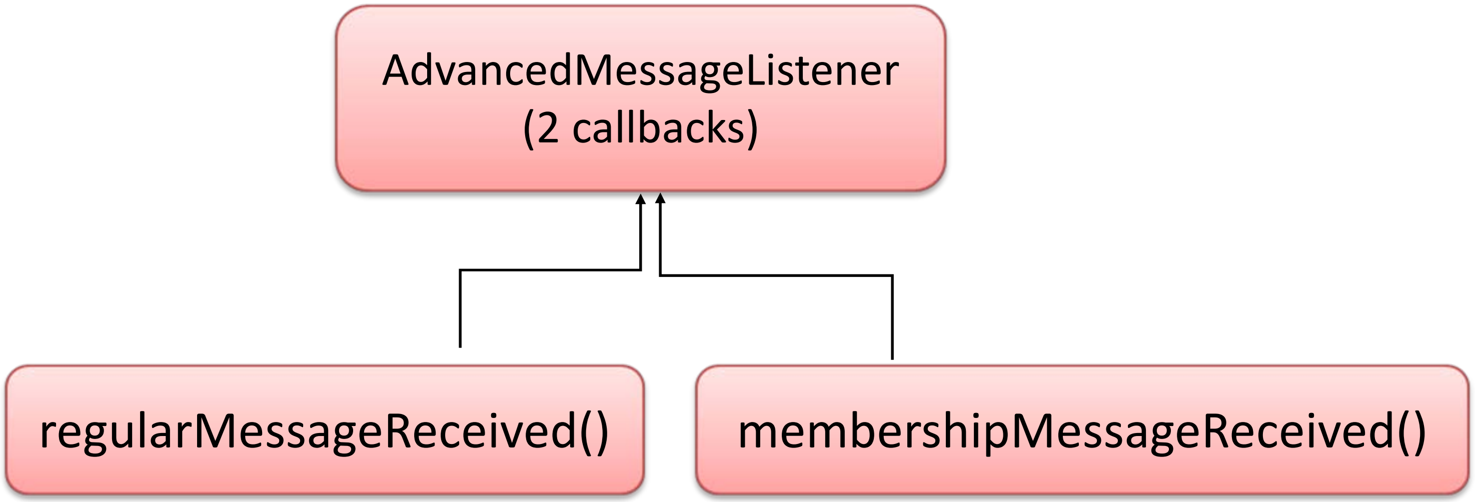
- An alternative way of receiving messages
- Interfaces:
  - BasicMessageListener
  - AdvancedMessageListener

```
connection.add(listener);
```

After being added to a connection, the listener will be alerted whenever a new message is received on the connection.



Listeners



- To remove a listener from the connection

```
connection.remove(listener);
```

Exceptions

- When an error occurs in a Spread method, a SpreadException is thrown
- Eg. **receive()** is called on a **SpreadConnection()** object before connect() is called on that object

- Any method that is declared as throwing a SpreadException must be placed within a try-catch block

```
try
{
    connection.multicast(message);
}
catch(SpreadException e)
{
    e.printStackTrace();
    System.exit(1);
}
```

References

- www.spread.org
- http://www.cnds.jhu.edu/