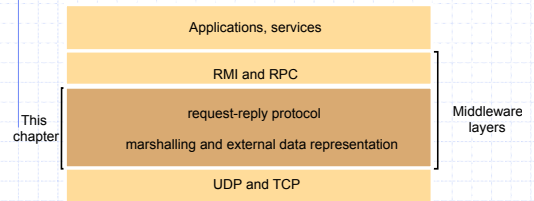


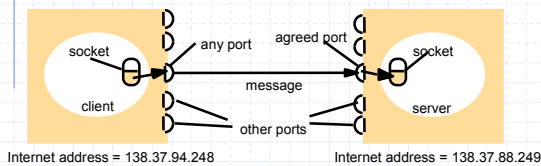
Distributed Objects: Communication and System Support

Ch.4,5 and 6

Middleware layers



Sockets and ports



Inter Process Communication

- ◆ IP address and port number. About 2^{16} ports are available for use by user processes.
- ◆ UDP and TCP abstraction of the above is a socket.
- ◆ Socket is an endpoint of communication between processes. Socket is associated with a protocol.
- ◆ IPC is transmitting a message between a socket in one process to a socket in another process.
- ◆ Messages sent to particular IP and port# can be received by the process whose socket is associated with that IP and port#.
- ◆ Processes cannot share ports with other processes within the computer. Can receive messages on diff ports.

Java API for networking

- ◆ java.net package supports for UDP and TCP communication.
- ◆ This package contains classes: DatagramPacket, DatagramSocket, ServerSocket, Socket and the associated methods.
- ◆ For example, DatagramSocket provides operations: send, receive, setSoTimeout, connect...

UDP client sends a message to the server and gets a reply

```

import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(String args[]){
        // args give message contents and server hostname
        try {
            DatagramSocket aSocket = new DatagramSocket();
            byte [] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByAddress(args[1]);
            int serverPort = 6789;
            DatagramPacket request = new DatagramPacket(m, args[0].length(), aHost,
                serverPort);

            aSocket.send(request);
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Reply: " + new String(reply.getData()));
            aSocket.close();
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
        } catch (IOException e){System.out.println("IO: " + e.getMessage());}
    }
}

```

UDP server repeatedly receives a request and sends it back to the client

```
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main(String args[]){
        try{
            DatagramSocket aSocket = new DatagramSocket(6789);
            byte[] buffer = new byte[1000];
            while(true){
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
        } catch (IOException e) {System.out.println("IO: " + e.getMessage());}
    }
}
```

9/24/2004

B.Ramamurthy

7

TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPCClient {
    public static void main (String args[]) {
        // arguments supply message and hostname of destination
        try{
            int serverPort = 7896;
            Socket s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream());
            DataOutputStream out =
                new DataOutputStream( s.getOutputStream());
            out.writeUTF(args[0]); // UTF is a string encoding see Sn 4.3
            String data = in.readUTF();
            System.out.println("Received: " + data);
            s.close();
        } catch (UnknownHostException e){
            System.out.println("Sock:" + e.getMessage());
        } catch (EOFException e){System.out.println("EOF:" + e.getMessage());}
        } catch (IOException e){System.out.println("IO:" + e.getMessage());}
    }
}
```

9/24/2004

B.Ramamurthy

8

TCP server makes a connection for each client and then echoes the client's request

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
        try{
            int serverPort = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true) {
                Socket clientSocket = listenSocket.accept();
                Connection c = new Connection(clientSocket);
            }
        } catch (IOException e) {System.out.println("Listen :"+ e.getMessage());}
    }
}

// this figure continues on the next slide
```

9/24/2004

B.Ramamurthy

9

(continued)

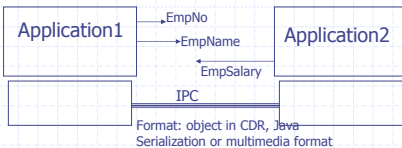
```
class Connection extends Thread {
    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;
    public Connection (Socket aClientSocket) {
        try {
            clientSocket = aClientSocket;
            in = new DataInputStream(clientSocket.getInputStream());
            out = new DataOutputStream(clientSocket.getOutputStream());
            this.start();
        } catch (IOException e) {System.out.println("Connection:" + e.getMessage());}
    }
    public void run(){
        try {
            // an echo server
            String data = in.readUTF();
            out.writeUTF(data);
            clientSocket.close();
        } catch (EOFException e) {System.out.println("EOF:" + e.getMessage());}
        } catch (IOException e) {System.out.println("IO:" + e.getMessage());}
    }
}
```

9/24/2004

B.Ramamurthy

10

External Data Representation and Marshalling



Applications can be CORBA applications, Java Applications or any other kind of application. Once We get out of the system space we need to follow Rules or protocols: CDR , java serialization, ior , ror are External data representation protocol. Objects can be passed by reference (CORBA) or by value (Java)

9/24/2004

B.Ramamurthy

11

External Data representation ... (contd.)

- ◆ An agreed standard for the representation of data structures and primitive values is called external data representation.
- ◆ Marshalling is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
- ◆ Unmarshalling is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.
- ◆ Two binary protocols: CORBA's Common Data Representation (CDR) and Java's Object Serialization.
- ◆ Two ASCII protocols: HTML (HTTP), XML

9/24/2004

B.Ramamurthy

12

CORBA CDR for constructed types

Type	Representation
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

9/24/2004

B.Ramamurthy

13

CORBA CDR message

index in sequence of bytes	← 4 bytes →	notes on representation
0-3	5	length of string
4-7	"Smit"	'Smith'
8-11	"h__"	
12-15	6	length of string
16-19	"Lond"	'London'
20-23	"on__"	
24-27	1934	unsigned long

The flattened form represents a

Struct Person { string name; string place; long year;};

struct with value: {'Smith', 'London', 1934}

9/24/2004

B.Ramamurthy

14

Indication of Java serialized form

Serialized values				Explanation
Person	8-byte version number		h0	class name, version number
3	int year	java.lang.String name:	java.lang.String place:	number, type and name of instance variables
1934	5 Smith	6 London	h1	values of instance variables

The true serialized form contains additional type markers; h0 and h1 are handles

Strings and characters are written out using UTF (Universal Transfer Format Reflection, an ability to inquire about the properties of the class makes The marshalling and unmarshalling functions quite generic, unlike CORBA Where CORBA compiler has to generate special operations for marshalling And unmarshalling.

9/24/2004

B.Ramamurthy

15

External Representation of a remote object reference

32 bits	32 bits	32 bits	32 bits	
Internet address	port number	time	object number	interface of remote object

Sample IOR: generated by Java-ORB application for a Stock Server Object.

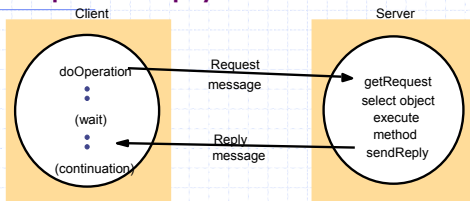
IOR:0000000000001b49444c3a53746f636b4f626a656374732f53746f636b3a312e30000000000001000000000000004000100000000017636173746f722e6373652e427566666616c6f2e45444550000994000000000018afabcafe00000025211cb86000000800000000000000000

9/24/2004

B.Ramamurthy

16

Request-reply communication



This is **reactive**.

How about **proactive**? **Push technology**. Server keeps sending messages to potential clients.

How about **P2P**? Peer to Peer if we have IORs and discovery protocol an we not do this?

9/24/2004

B.Ramamurthy

17

Operations of the request-reply protocol

`public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)` sends a request message to the remote object and returns the reply.

The arguments specify the remote object, the method to be invoked and the arguments of that method.

`public byte[] getRequest ();`

acquires a client request via the server port.

`public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);` sends the reply message reply to the client at its Internet address and port.

9/24/2004

B.Ramamurthy

18

Request-reply message structure

messageType	<i>int (0=Request, 1= Reply)</i>
requestId	<i>int</i>
objectReference	<i>RemoteObjectRef</i>
methodId	<i>int or Method</i>
arguments	<i>array of bytes</i>

9/24/2004

B.Ramamurthy

19

HTTP: An Example for Request/Reply Protocol

- ◆ Web servers manage resources implemented in different ways:
 - As data: text of HTML page, an image or class of an applet
 - As a program: cgi programs and servlets that can be run on the web server.
- ◆ HTTP protocol supports a fixed set of methods: GET, PUT, POST, HEAD, DELETE etc see p.152.
- ◆ In addition to invoking methods on the resources, the protocol allows for content negotiation (EX: frame, text, printable etc.) and password authentication.

9/24/2004

B.Ramamurthy

20

HTTP request message

<i>method</i>	<i>URL or pathname</i>	<i>HTTP version</i>	<i>headers</i>	<i>message body</i>
GET	//www.dcs.qmw.ac.uk/index.htm	HTTP/1.1		

9/24/2004

B.Ramamurthy

21

HTTP reply message

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

9/24/2004

B.Ramamurthy

22

Group Communications

- ◆ Pairwise exchange of messages is not the best model for communications from one process to a group of processes.
- ◆ A multicast is an operation that sends a single message from one process to each member of a group of processes.
- ◆ Issues: fault-tolerance, **discovery of service in a spontaneous networking environment**, better performance thru replicated data, **propagation of event notification**.

9/24/2004

B.Ramamurthy

23

Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
    public static void main(String args[]){
        // args give message contents & destination multicast group (e.g. "228.5.6.7")
        try {
            InetAddress group = InetAddress.getByName(args[1]);
            MulticastSocket s = new MulticastSocket(6789);
            s.joinGroup(group);
            byte [] m = args[0].getBytes();
            DatagramPacket messageOut =
                new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);
        }
    }
}
```

// this figure continued on the next slide

9/24/2004

B.Ramamurthy

24

...continued

```
// get messages from others in group
byte[] buffer = new byte[1000];
for(int i=0; i< 3; i++) {
    DatagramPacket messageIn =
        new DatagramPacket(buffer, buffer.length);
    s.receive(messageIn);
    System.out.println("Received:" + new String(messageIn.getData()))
}
s.leaveGroup(group);
}catch (SocketException e){System.out.println("Socket: " + e.getMessage());}
}catch (IOException e){System.out.println("IO: " + e.getMessage());}
}
```

<http://www.weblogic.com/docs51/examples/ejb/basic/statelessSession/index.html>

9/24/2004

B.Ramamurthy

25

Sockets used for datagrams

Sending a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
• bind(s, ClientAddress)
•
• sendto(s, "message", ServerAddress)
```

Receiving a message

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
• bind(s, ServerAddress)
•
• amount = recvfrom(s, buffer, from)
```

ServerAddress and ClientAddress are socket addresses

9/24/2004

B.Ramamurthy

26

Sockets used for streams

Requesting a connection

```
s = socket(AF_INET, SOCK_STREAM, 0)
•
•
• connect(s, ServerAddress)
•
•
• write(s, "message", length)
```

Listening and accepting a connection

```
s = socket(AF_INET, SOCK_STREAM, 0)
•
• bind(s, ServerAddress);
• listen(s,5);
•
• sNew = accept(s, ClientAddress);
•
• n = read(sNew, buffer, amount)
```

ServerAddress and ClientAddress are socket addresses

9/24/2004

B.Ramamurthy

27