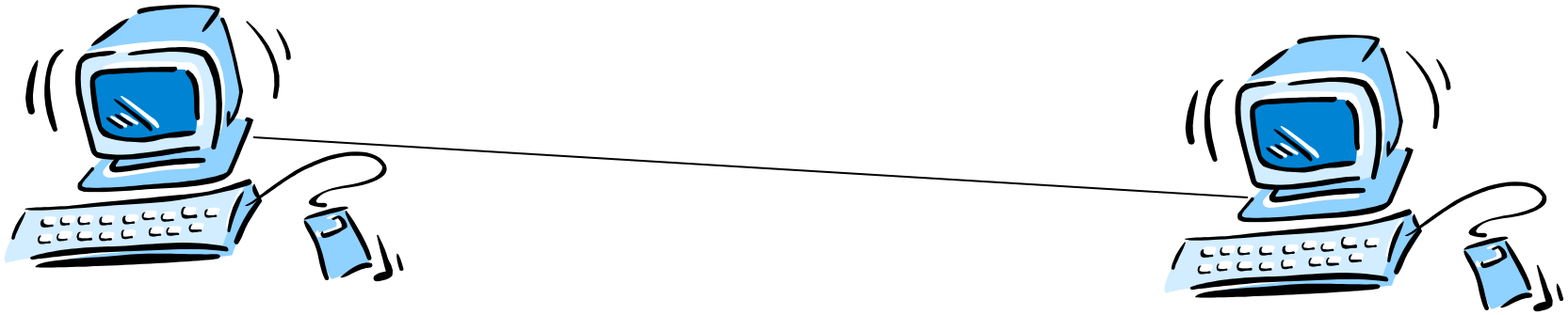
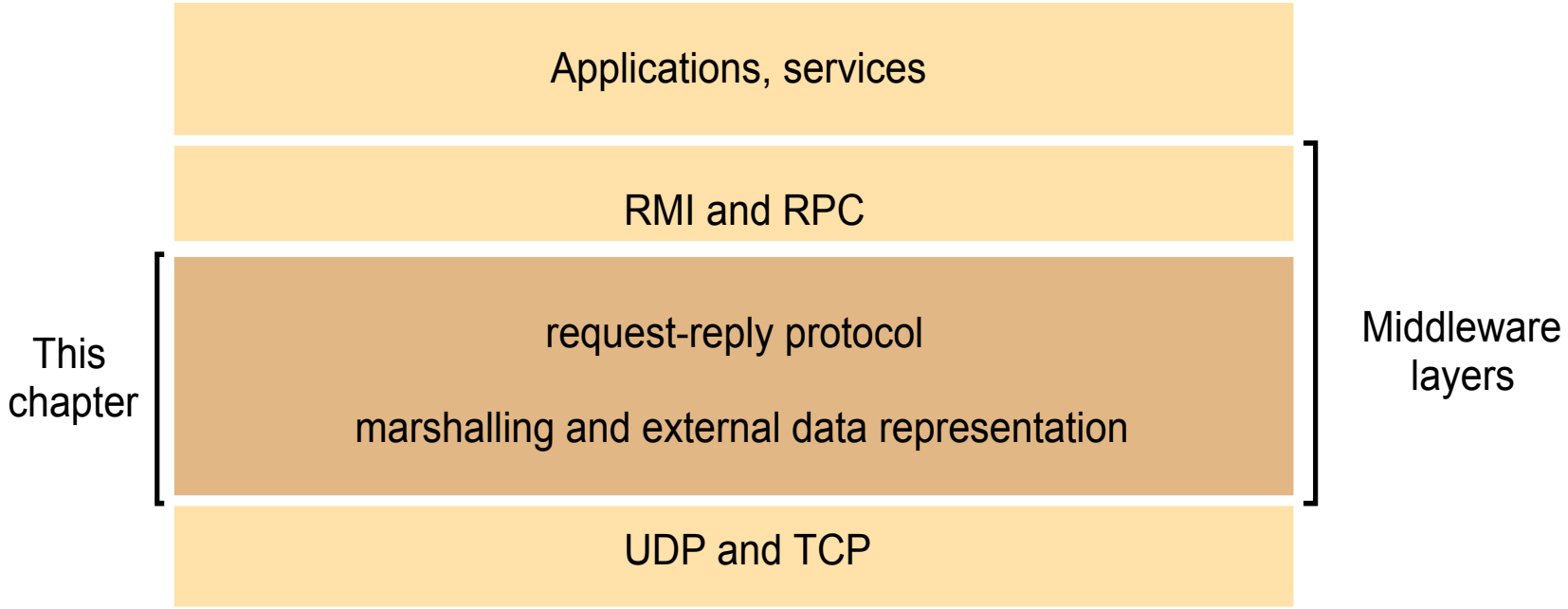


Organizational Communications and Distributed Object Technologies

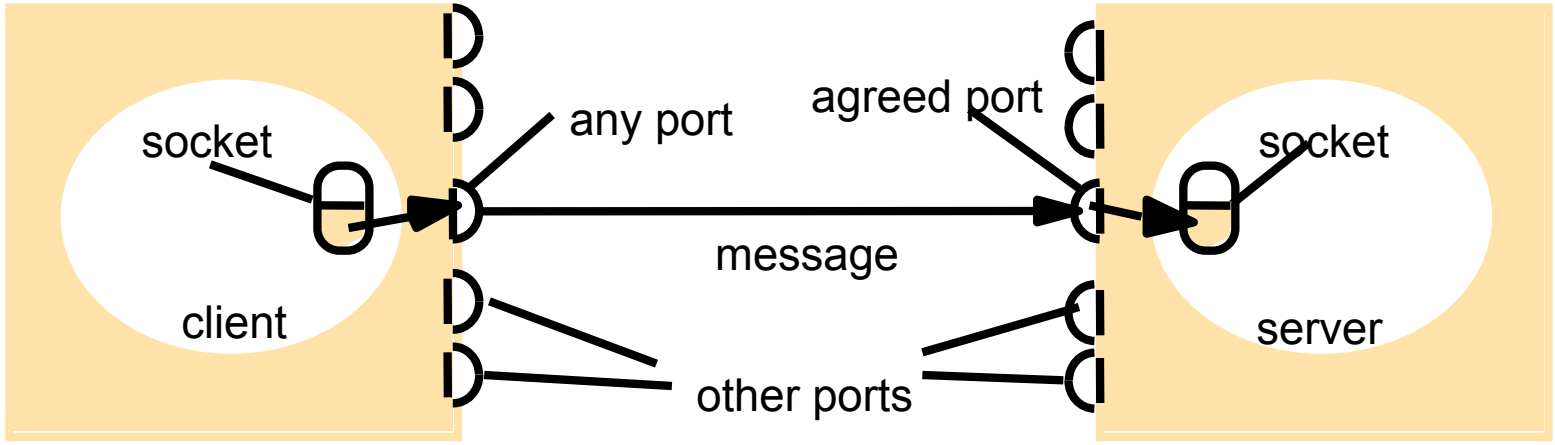
Lecture 8

Chapter 4: Inter-process Communications

Middleware layers



Socket and Port Abstractions



Internet address = 138.37.94.248

Internet address = 138.37.88.249

A UDP Client

```
import java.net.*;
import java.io.*;
public class UDPClient{
   public static void main(String args[]){
            // args give message contents and server hostname
            DatagramSocket aSocket = null;
             try {
                        aSocket = new DatagramSocket();
                        byte [] m = args[0].getBytes();
                        InetAddress aHost = InetAddress.getByName(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()));
             }
             catch (Exception e)
             {
                 System.out.println("Problem: " + e.toString());
             }
             finally {
                 if(aSocket != null) aSocket.close();
             }
   }
}
```

A UDP Server

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

Two Demonstrations

1) Run the UDP client and server on the same machine.

   /Users/mm6/mm6/mm6/www/95-702/UDPNetworking
   java UDPServer
   java UDPClient hello localhost

2) Run a UDP client on an Android device. The server will run on a laptop. In this case, the UDP server will accept arithmetic expressions.

   Netbeans 6.8: Homework3Part1UDPProject/UDPServer.java
   Eclipse: AndroidUDPCalculatorProject

Quiz:

What if the client sends a packet and that packet is lost?
Does this server handle concurrent visitors?
Is the packet safe from eavesdropping?
Could we visit this server with a .Net client?

TCP Client

```
import java.net.*;
import java.io.*;
public class TCPClient {

        public static void main (String args[]) {
                // arguments supply message and hostname of destination
                Socket s = null;
                    try{
                                int serverPort = 7896;
                                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) ;
                    }
                     catch (Exception e) {
                                System.out.println("Trouble: " + e.getMessage());
                    }
                    finally {
                          if(s!=null) try {s.close();}
                            catch (IOException e) {
                                System.out.println("close:"+e.getMessage());
                            }
            }
        }
}
```

TCP Server(1)

```
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();
                        System.out.println("Got connection");
                        Connection c = new Connection(clientSocket);
                  }
            }
            catch(IOException e) {
               System.out.println("Listen :"+e.getMessage());
     }
   }
}
```

TCP Server(2)

```
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 {
                    String data = in.readUTF();
                    out.writeUTF("From server: " + data);
            }
            catch(Exception e) {
                System.out.println("EOF:"+e.getMessage());
            }
            finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
```

Demonstration

/Users/mm6/mm6/mm6/www/95-702/TCPNetworking

java TCPServer
java TCPClient hello localhost

Quiz

What if the client sends a packet and that packet is lost?
Does this server handle concurrent visitors?
Is the packet safe from eavesdropping?
Could we visit this server with a .Net client?

External Data Representation and Marshalling

Messages consist of sequences of bytes.

Interoperability Problems

- Big-endian, little-endian byte ordering
- Floating point representation
- Character encodings (ASCII, UTF-8, Unicode, EBCDIC)

So, we must either:

- Have both sides agree on an external representation or transmit in the sender's format along with an indication of the format used. The receiver converts to its form.

External Data Representation and Marshalling

External data representation – an agreed standard for the representation of data structures and primitive values

Marshalling – 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 into an equivalent representation at the destination

The marshalling and unmarshalling are intended to be carried out by the middleware layer

External Data Representation and Marshalling

Quiz:

If, in the TCPNetworking example, we passed java objects rather than simple characters, would the server interoperate with a .NET client?

Three Important Approaches

To External Data Representation and Marshalling:

CORBA's CDR binary data may be used by different programming languages

Java and .Net Remoting Object Serialization are both platform specific (that is, Java on both sides or .Net on both sides) and binary.

XML is a textual format, verbose when compared to binary but more interoperable.

Interoperability

Consider int j = 3;

What does it look like in memory?
00000000000000000000000000000011

How could we write it to the wire?

Little-Endian approach	Big-Endian Approach
Write 00000011	Write 0000000
Then 00000000	Then 0000000
Then 00000000	Then 0000000
Then 00000000	Then 0000011

The receiver had better know which one we are using!

Binary vs. Unicode

Consider int j = 3;

j holds a binary representation 00…011

We could also write it in Unicode.

The character ‘3’ is coded as 0000000000110011

Binary is better for arithmetic.

The character ‘Ω’ is coded as 0000001110101001

The number 43 can be written as a 32 bit binary integer or as two 16 bit Unicode characters

The receiver had better know which one we are using!

Let's Examine Three Approaches

- CORBA
- Java
- XML

CORBA Common Data Representation (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

- Can be used by a variety of programming languages.
- The data is represented in binary form.
- Values are transmitted in sender's byte ordering which is specified in each message.
- May be used for arguments or return values in RMI.

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*

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

In CORBA, it is assumed that the sender and receiver have common knowledge of the order and types of the data items to be transmitted in a message.

CORBA

CORBA Interface Definition Language (IDL)


```
struct Person {
    string name;
    string place;
    long year;
};
```

generates

Appropriate marshalling
and unmarshalling operations

Java

```
public class Person implements Serializable {
   private String name;
   private String place;
   private int year;
   public Person(String nm, place, year) {
      nm = name; this.place = place; this.year =
    year;
   }
   // more methods
}
```

Java Serialization

Serialization refers to the activity of flattening an object or even a connected set of objects

- May be used to store an object to disk
- May be used to transmit an object as an argument or return value in Java RMI
- The serialized object holds Class information as well as object instance data
- There is enough class information passed to allow Java to load the appropriate class at runtime. It may not know before hand what type of object to expect

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 are references to other locations within the serialized form
The above is a binary representation of {'Smith', 'London', 1934}

XML

```
<p:person p:id="123456789" xmlns:p="http://www.andrew.cmu.edu/~mm6">
    <p:name>Smith</p:name>
    <p:place>London</p:place>
    <p:year>1934</p:year>
</p:person>
```

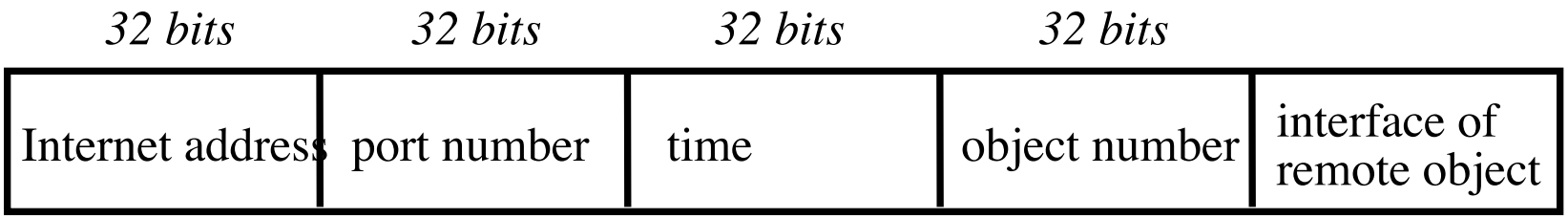
- Textual representation is readable by editors like Notepad or Textedit.
- But can represent any information found in binary messages.
- How? Binary data (e.g. pictures and encrypted elements) may be represented in Base64 notation.
- Messages may be constrained by a grammar written in XSD.
- An XSD document may be used to describes the structure and type of the data.
- Interoperable! A wide variety of languages and platforms support the marshalling and un-marshalling of XML messages.
- Verbose but can be compressed.
- Standards and tools still under development in a wide range of domains.

Passing Pointers

In systems such as Java RMI or CORBA or .NET remoting, we need a way to pass pointers to remote objects.

Quiz: Why is it not enough to pass along a heap address?

Representation of a Remote Object Reference

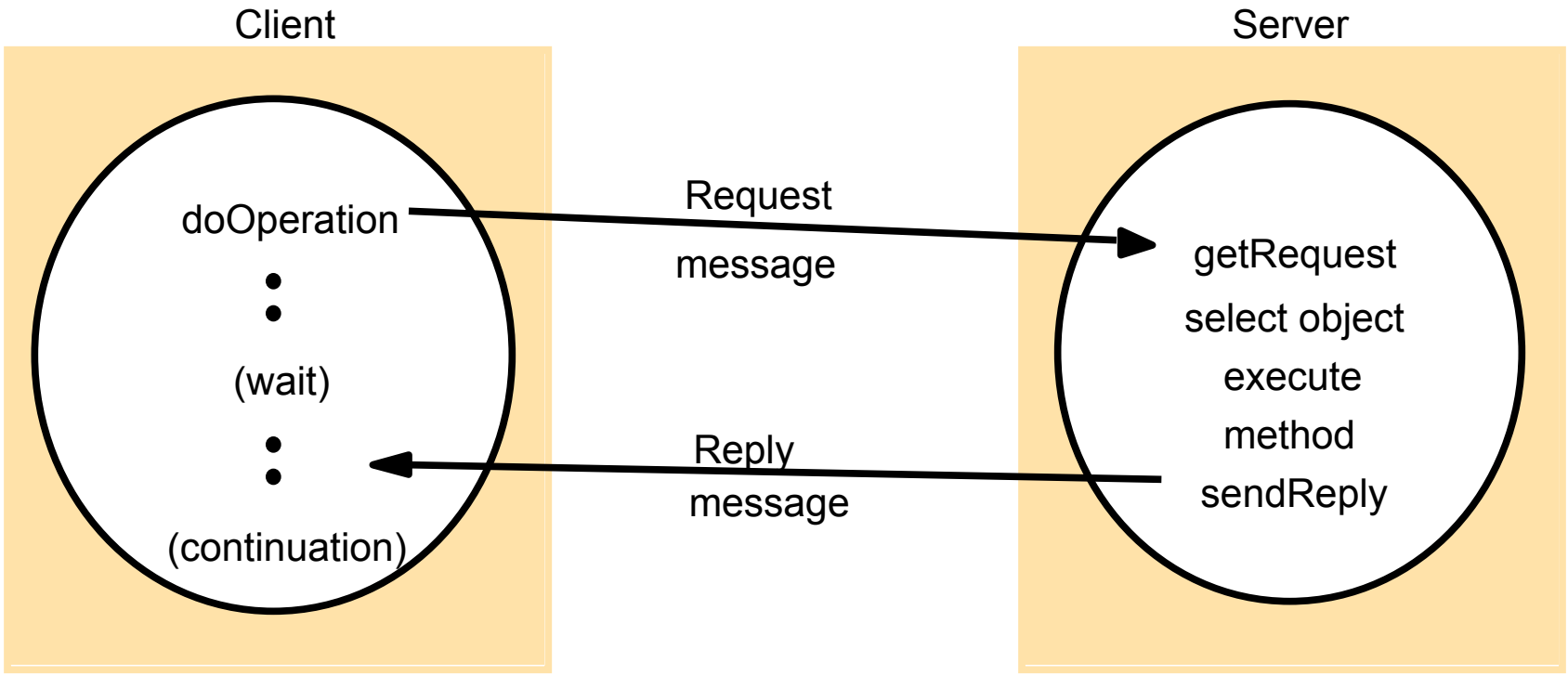


A remote object reference is an identifier for a remote object.
May be returned by or passed to a remote method in Java RMI.

A Request Reply Protocol

OK, we know how to pass messages and addresses of objects.
But how does the middleware carry out the communication?

UDP Style Request-Reply Communication



UDP Based Request-Reply Protocol

Client side
b = doOperation

Client side:

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.

Server side:

Server side:

b=getRequest()
operate
sendReply()

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.

Failure Model of UDP Request Reply Protocol

Client side
b = doOperation

A UDP style doOperation may timeout while waiting.

What should it do?

-- return to caller passing an error message

-- but perhaps the request was received and the response was lost, so, we might write the client to try and try until convinced that the receiver is down

In the case where we retransmit messages the server may receive duplicates

Server side:

b=getRequest()
operate
sendReply()

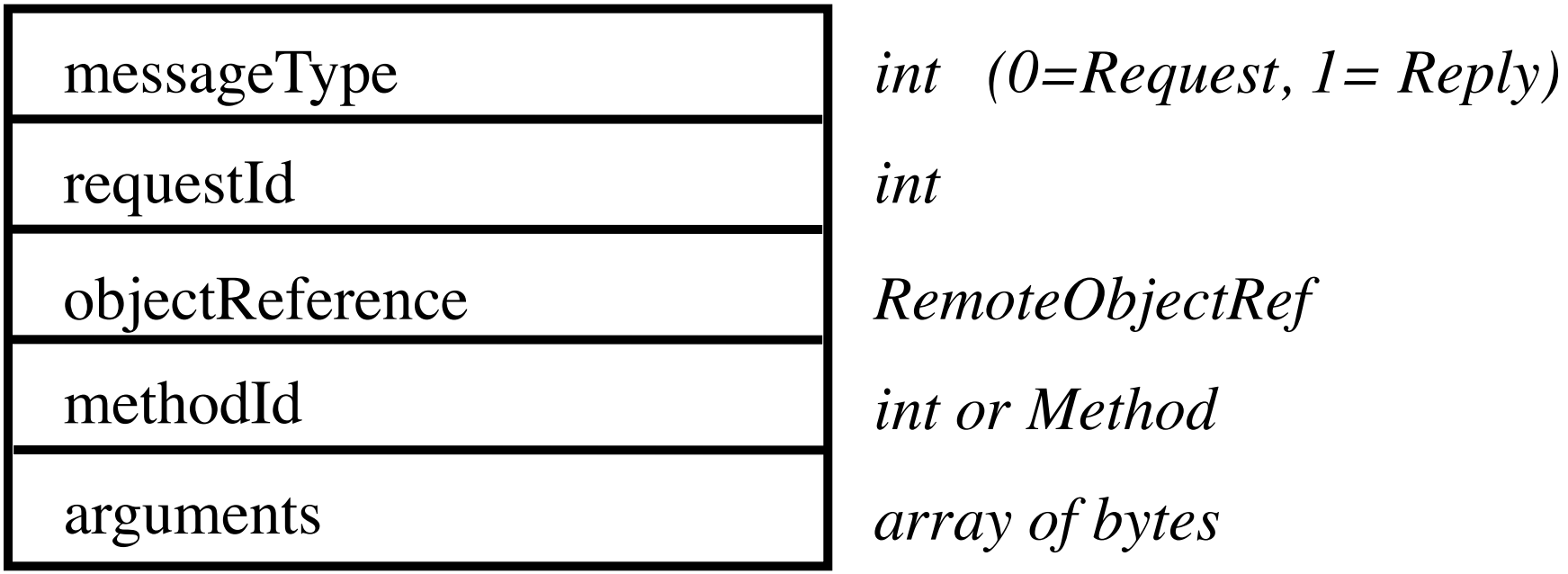
Failure Model Handling Duplicates (Appropriate for UDP but not TCP)

- Suppose the server receives a duplicate messages.
- The protocol may be designed so that either

  (a) it re-computes the reply (in the case of idempotent operations) or

  (b) it returns a duplicate reply from its history of previous replies
- Acknowledgement from client clears the history

Request-Reply Message Structure



RPC Exchange Protocols Identified by Spector[1982]

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

R = no response is needed and the client requires no confirmation

RR= a server's reply message is regarded as an acknowledgement

RRA= Server may discard entries from its history

A Quiz

Why is TCP chosen for request-reply protocols?

Variable size parameter lists.
TCP works hard to ensure that messages are delivered reliably.
So, no need to worry over retransmissions, filtering of duplicates or histories.
The middleware is easier to write.

HTTP Request Message

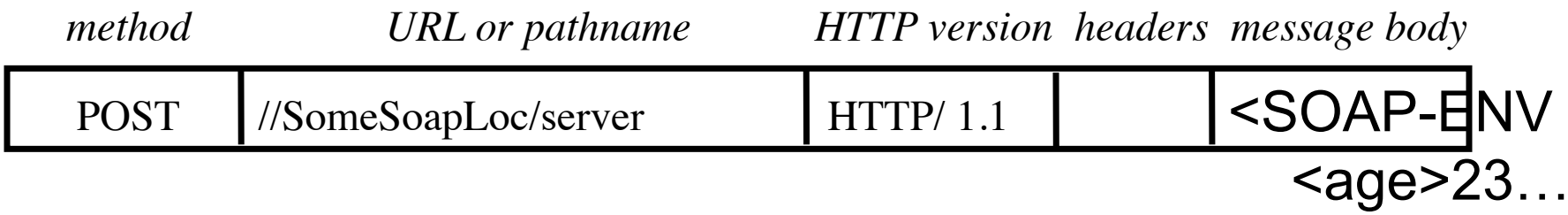
Traditional HTTP request

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.SomeLoc/?age=23	HTTP/ 1.1		

HTTP Is Implemented over TCP.

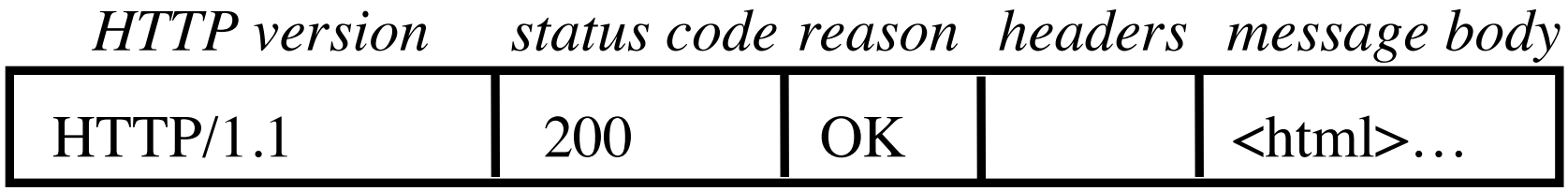
HTTP SOAP Message

Web Services style HTTP request

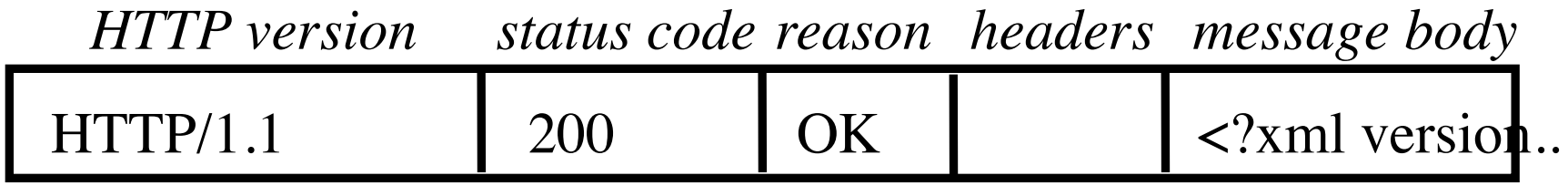


HTTP is extensible.

Traditional HTTP Reply Message



HTTP Web Services SOAP Reply Message



A Working Toy Example

Server side code:

servant MyCoolClassServant.java
server CoolClassServer.java
skeleton MyCool_Skeleton.java
interface MyCoolClass.java

Client side code:

Client CoolClient.java
Interface MyCoolClass.java
stub CoolClass_Stub.java

Netbeans 6.8
LowLevelDistributedObjectProject
LowLevelDistributedObjectProjectClient

CoolClassServer.java

```
public class CoolClassServer {


    public static void main(String args[]) {

        System.out.println("Main");

        MyCool_Skeleton cs =
                        new MyCool_Skeleton(new MyCoolClass_Servant());

        cs.serve();

    }
}
```

MyCoolClass_Servant.java

```
public class MyCoolClass_Servant implements MyCoolClass {

    private String n[] = {"printer","stereo","TV","ipod","pda"};

    private String a[] = {"HP200XT","Kenwood200","Panasonic","Apple","Palm"};

    public String getDevice(String name) {

       for(int i = 0; i < n.length; i++) {
            if(n[i].equals(name)) return a[i];
       }
       return "No device";
    }
}
```

MyCool_Skeleton.java (1)

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
import java.net.ServerSocket;

public class MyCool_Skeleton {

  MyCoolClass mcc;

  public MyCool_Skeleton(MyCoolClass p) {

     mcc = p;
  }
```

MyCoolSkeleton.java (2)

```
public void serve()   {
 try {
        ServerSocket s = new ServerSocket(9000);
        while(true) {
          Socket socket = s.accept();
          ObjectInputStream i = new
                  ObjectInputStream(socket.getInputStream());
          String name = (String)i.readObject();
          String result = mcc.getDevice(name);
          ObjectOutputStream o = new
                  ObjectOutputStream(socket.getOutputStream());
          o.writeObject(result);
          o.flush();
        }
    }
    catch(Throwable t) {
            System.out.println("Error " + t);
            System.exit(0);
    }
}
```

MyCoolClass.java

```
// Exists on both the client and server


public interface MyCoolClass {

    public String getDevice(String name) throws Exception;
}
```

CoolClient.java

```
public class CoolClient {

    public static void main(String args[]) {

      try {

            MyCoolClass p = new CoolClass_Stub();
            System.out.println(p.getDevice(args[0]));
      }
      catch(Throwable t) {
          t.printStackTrace();
          System.exit(0);
      }
    }
}
```

CoolClass_Stub.java (1)

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;

public class CoolClass_Stub implements MyCoolClass {

  Socket socket;
  ObjectOutputStream o;
  ObjectInputStream i;
```

CoolClass_Stub.java (2)

```
public String getDevice(String name) throws Exception {

    socket = new Socket("localhost",9000);
    o = new ObjectOutputStream(socket.getOutputStream());
    o.writeObject(name);
    o.flush();

    i = new ObjectInputStream(socket.getInputStream());

    String ret = (String)(i.readObject());
    socket.close();
    return ret;
  }
}
```

Discussion

With respect to the previous system, let's discuss:

Request-Reply protocol.
Marshalling and external data representation.
Interoperability.
Security.
Reliability.
Performance.
Openness.
Use of Metadata.
Remote references.