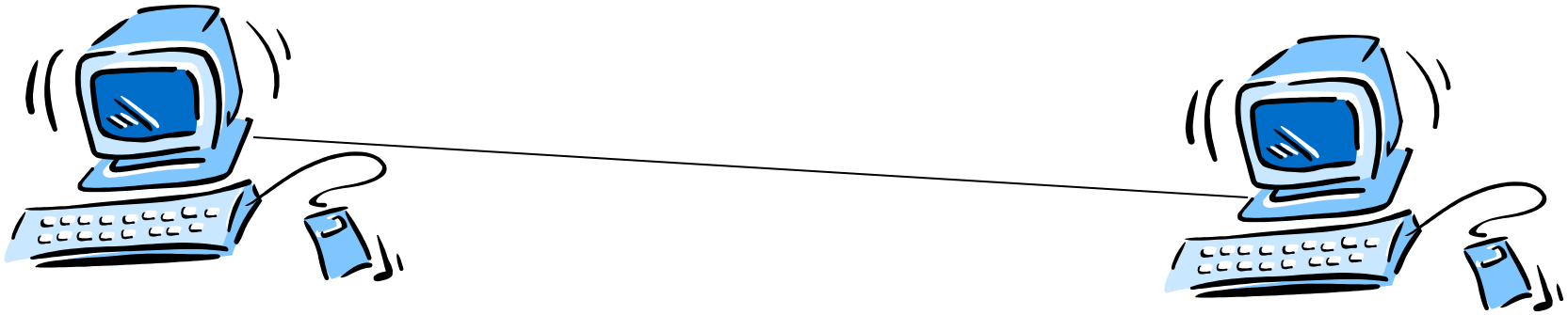


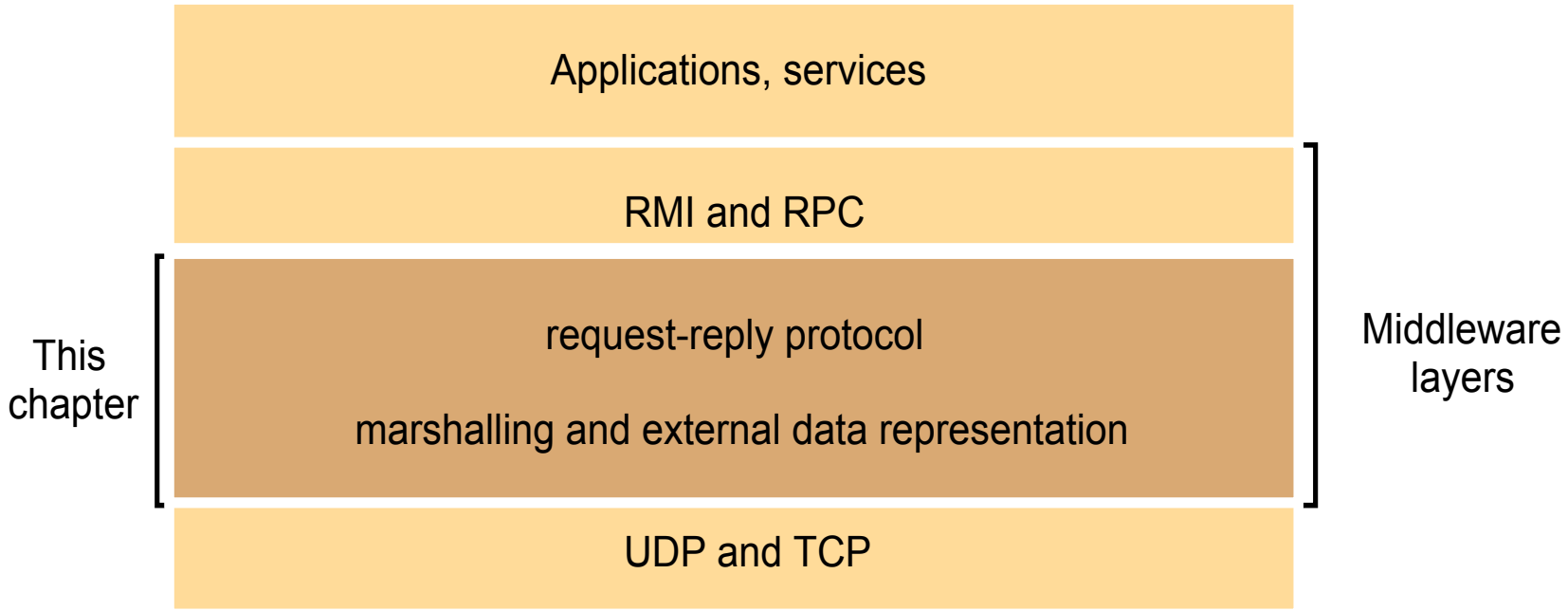
95-702 Distributed Systems

Chapter 4: Inter-process Communications

Objectives

- Understand the purpose of middleware.
- Understand how external data representations contribute to interoperability.
- Understand how external data representations contribute to speed.
- Understand marshalling/unmarshalling
- Understand CORBA's CDR
- Understand Java's serialization
- Understand XML and JSON
- Understand how remote object references may be represented.
- A UDP based request response protocol
- Failure models
- Discussion questions

Middleware layers



Middleware provides a higher level programming abstraction for the development of distributed systems. (Coulouris text).

Moving values around on a network

Passing values over a network may be problematic. Why?
If both sides are the same (homogenous), no problem.
But if the two sides differ in the way they represent data then we are faced with interoperability problems:

1. Big-endian, little-endian byte ordering may differ
2. Floating point representation may differ
3. Character encodings (ASCII, UTF-8, Unicode, EBCDIC) may differ as well.

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.

Quiz: Which one of these approaches are we using in class today?
Quiz: Which one of these approaches is used on WWW?

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 usually carried out by the middleware layer

External Data Representation and Marshalling

Quiz:

Suppose we write a TCP server in C++.
Could we open a Java TCP connection to the server?
Suppose we write a client in Java that sends a Java object to the server.
Would the content of the Java object be reconstructed into C++?

Interoperability concern: Big/Little Endian

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!

Interoperability concern: 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
CPU’s like binary for integer 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!

Three Important Approaches to external data representation

CORBA's CDR (Common Data Representation) 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 and slow when compared to binary but interoperable. JSON is like XML but more compact.

Three important approaches to external data representation

- CORBA's Common Data Representation
  Both sides have the IDL beforehand. This is similar to Google's protocol buffers.
  Quiz: What does an IDL buy us?
- Java's serialization
  Use Java serialization to marshal and un-marshal to a network or to storage. No IDL used.
- Web Service use of XML or JSON. In the case of XML, XSDL or WSDL may act as an IDL.

CORBA in a Nutshell

- From the Object Management Group (OMG) around since the late 80's
- OMG an international, open membership, not-for-profit technology standards group
- The CORBA effort was all about distributed objects on heterogeneous platforms.
- CORBA does a lot of things but central is the idea of passing around objects by value and references to objects.
- CORBA 2.0 uses CDR to represent all of the datatypes that may be passed as arguments to or return values from a method.

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.

Example 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

One can easily include the proxy code and make calls to its methods.

Another approach: Java Serialization

```
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}

Web Service use of XML

```
<p:person  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>
```

- How does the web work? (Text or binary?) (Compact messages?)
- Textual representation is readable by editors like Notepad or Textedit. We still need an agreement on what character encoding to use, e.g., an HTTP header might say Content-Type: text/xml; charset:ISO-8859-1;
- 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 XSDL.
- An XSDL 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. (Compare with CORBA or Java serialization.)
- Verbose and slow
- Standards and tools still under development in a wide range of domains.

Web Service use of JSON

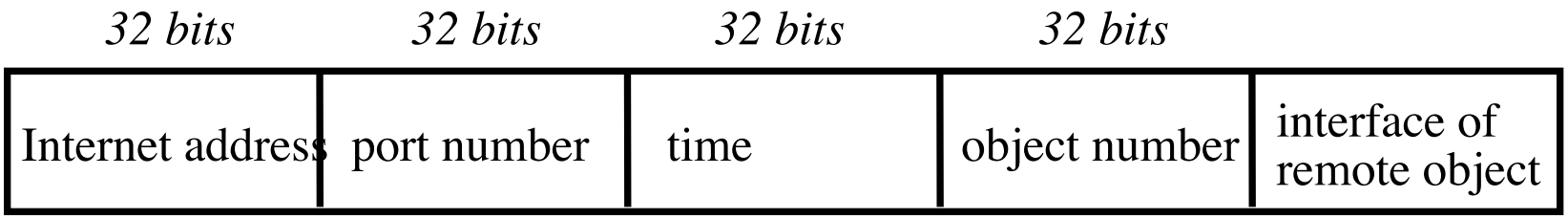
```
{ “person” : {  “name” : “Smith”
                “place”:”London”
                “year”:”1934”}
}
```

- Textual representation is readable by editors like Notepad or Textedit. UTF-8 is the standard encoding.
- 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 are constrained by a general grammar, see www.JSON.org
- Interoperable! A wide variety of languages and platforms support the marshalling and un-marshalling of JSON messages.
- The de-facto standard in many RESTful applications.

In distributed OOP, we need to pass pointers…

- In stand alone OOP, we use pointers all the time.
  BigInteger x = new BigInteger();
- We are pointing to objects that live on the heap.
- In systems such as Java RMI or CORBA or .NET remoting, we need a way to pass pointers to remote objects.
  We want x to point to an object living on some distant machine
- Quiz: Why is it not enough to pass along a heap address?
- Note: With web services we may make good use of URL's - BUT we are not trying to build distributed OOP.

Representation of a Remote Object Reference



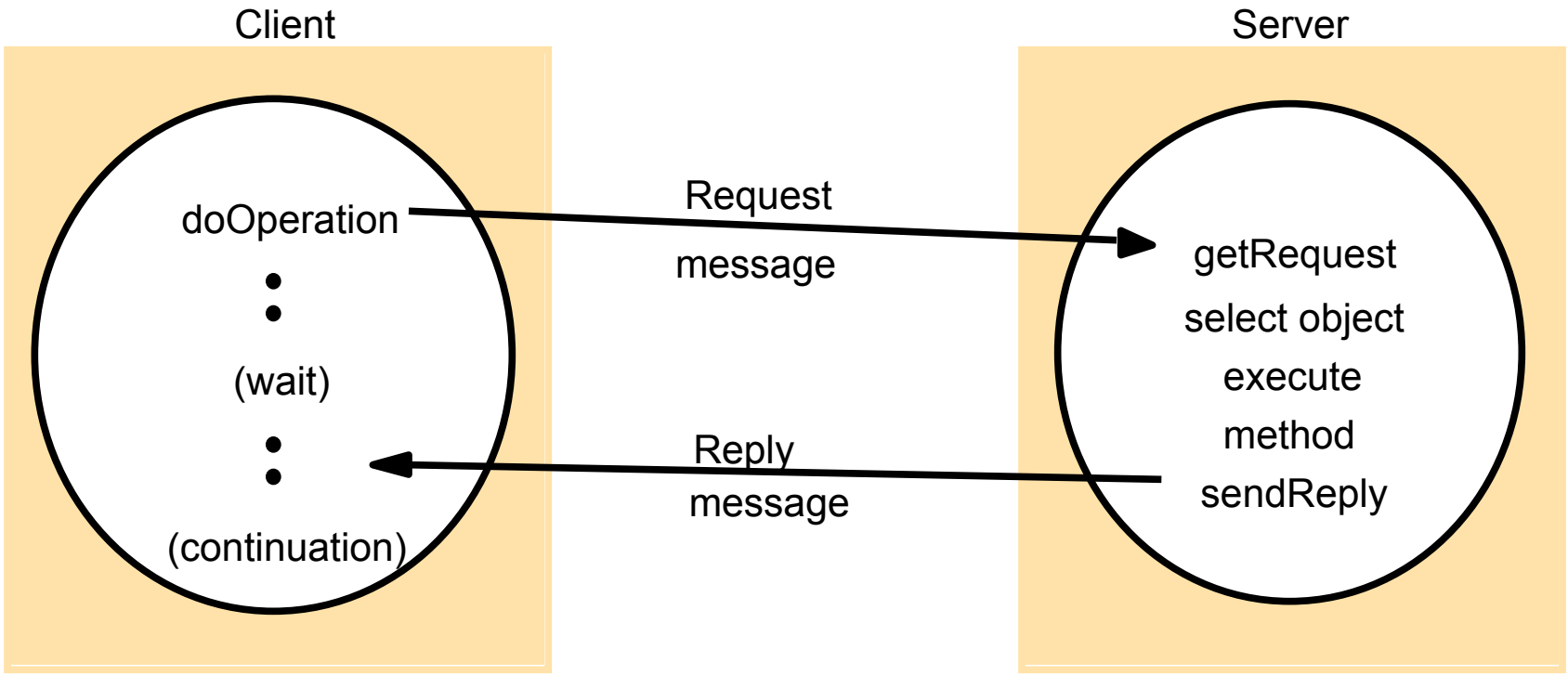
A remote object reference is an identifier for a remote object.
May be returned by or passed to a remote method in Java RMI.

How do these references differ from local references?

A Request Reply Protocol

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

A UDP Style Request-Reply Is Possible



UDP Based Request-Reply Protocol

Client side
b = doOperation(r,2,b)

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()
coolOperation
sendReply(re,ch,cp)

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()

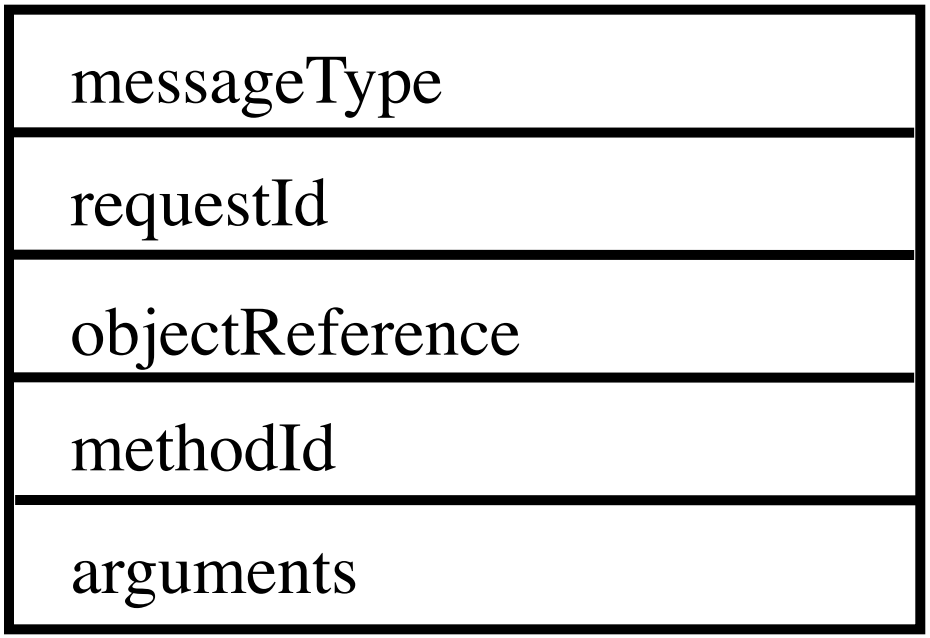
Failure Model for Handling Duplicates

- 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
- An acknowledgement from the client may be used to clear the history

Request-Reply Message Structure



int (0=Request, 1= Reply)

int

RemoteObjectRef

int or Method

array of bytes

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

Discussion

Compare and contrast web services with distributed object approaches in terms of the following:

- Marshalling and external data representation
- Interoperability
- Security
- Reliability
- Performance
- Remote references
- Full OOP
- Describe how the protocols of the internet allow for heterogeneity.
- Describe how middleware allows for heterogeneity.