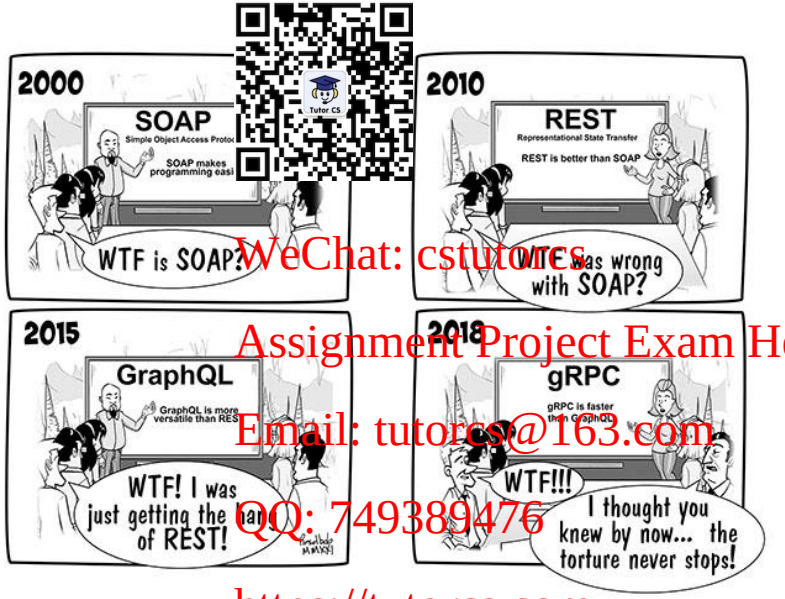
RPC RMI SOAP

Lecturer: Dr. Joseph Doyle

Introduction



https://devops.com/the-torture-never-stops/

Introduction

- When distributed systems were first utilised processes communicated by passing messages
- Messages only contained data
- The interpretation of this message had to be agreed between the sender and receiver
- This makes it difficult to reuse components and allow interoperability between distributed systems

Review: Sockets

Socket: door between application process and end-end-transport protocol

The API for applications to communicate across the network

Review: Sockets and ports

- Logical resources managed by the operating system
- **Sockets** are always **assigned a free port** when created
- Each process on a networked host can be addressed remotely by the port number it is listening to
- TCP and UDP ports are independent
- For **server-side** applications, **default port numbers** are defined
  - HTTP -> 80
  - HTTPS -> 443
  - SMTP -> 25

Review: Socket Server Example


```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Review: Socket Client Example


```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Introduction

- Technologies were introduced (Beginning with Remote Procedure Call (RPC)) to make communication between distributed processes more uniform, reusable and user friendly
- Essentially these technologies allow users to call functions on different physical machines as if they were local processes

Remote Procedure Call (RPC)

- Proposed in the 1970s, first practically implemented in the early 1980s
- Earliest popular implementation on Unix system was Sun's RPC in 1984 which was used to support the Network File System (NFS)
- Popular implementations today include
  - XML-RPC
  - JSON-RPC

Remote Procedure Call (RPC)

- Goal of RPC is to allow cli[illegible] different physical machine to call procedures as if they wer[illegible] to the local machine
- This is abstracted from the user so they make remote procedure calls in the exact same way that they make local calls
- The RPC implementation is responsible for
  - Connecting to the remote host
  - Sending the parameters
  - Performing the operation on the remote host
  - Returning the results

Characteristics of RPC

- A familiar interface for application developers
- Allow the implementation of the request reply paradigm
- Includes a standard message format
- Includes a standard interface to allow the reuse of code

Limitations of RPC

- Calls by reference are impossible as both machines use a different address space
- Large object (classes) which might normally be passed by reference need to be copied
- There may be byte ordering issues (Big Endian vs Little Endian)
- There may be formatting issues (ASCII, UTF-8, UTF-16, UTF-32)

Big Endian Vs Little Endian

https://en.wikipedia.org/wiki/Endianness

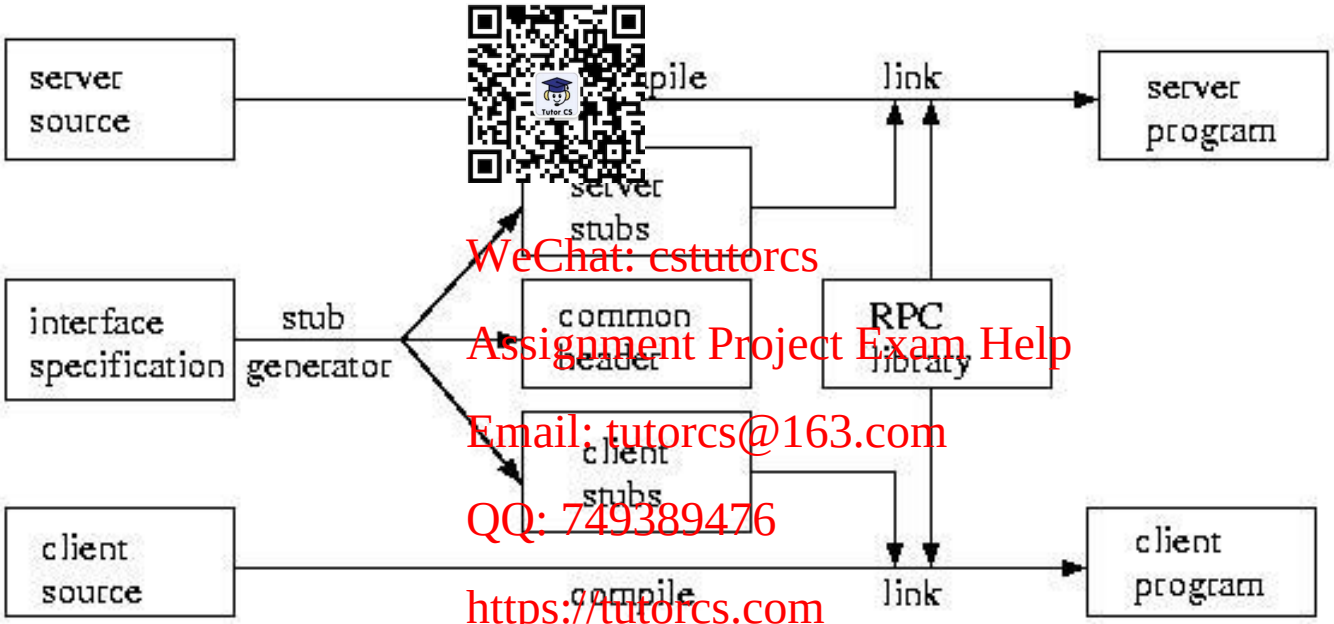
RPC Procedure

- RPC is a request response protocol
- The procedure is initiated by a client who sends a request message to a known remote server to execute a specified function with the supplied parameters
- The remote server sends a response to the client and the procedure continues
- The client will wait for the response from the server unless an asynchronous request message is sent to the server

RPC Events

- The events associated with a RPC are as follows
  - The client contacts the client stub. As this is a local call parameters are pushed to the stack in the normal way
  - The client stub packs the parameters into a message. This is known as marshalling. The client stub then makes a system call to send the message
  - The client's local operating system sends the message to the server
  - The server's local operating system passes the message to the server stub
  - The server stub unpacks the parameters of the message. This is known as unmarshalling
  - The server stub calls the remote procedure. The reply uses the same procedure in reverse

Compilation



https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture3.html

RPC Stub

- The stub is a gateway to distributed system objects and all outgoing requests to server side objects that are routed through it
- It also includes network logic to ensure reliable communication between the client and the server
- It is responsible for:
  - Initiating communication with the server
  - Marshalling and unmarshalling messages
  - Informing the server that the procedure should be called

Marshalling

- Marshalling is the process of transforming the memory representation of an object into a format suitable for storage or transmission
- Parameters sent in an RPC call must be marshalled before they can be sent to the remote procedure
- Marshalling is also used in the .NET framework and in the Mozilla Application Framework

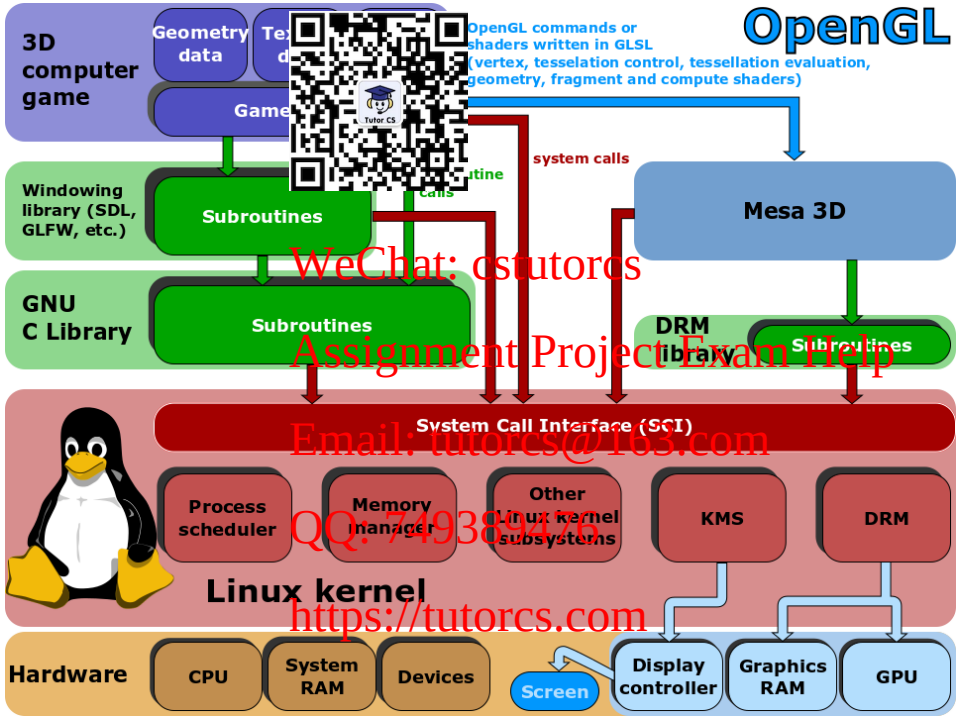
Marshalling Example

- Consider a program which works in user space and kernel space
- To transition from user space to kernel space a system call is required
- This is a slow operation which can take microseconds to complete so the number of system calls should be minimised
- To minimise the number of system calls a buffer of commands can be maintained in user space and in kernel space

Marshalling Example (cont)

- Commands waiting for execution are marshalled into the user space buffer
- When the kernel space buffer is nearly empty a system call is executed and commands in the user space buffer are transferred into the kernel space buffer
- This approach used in Linux's OpenGL to minimise system calls when rendering

Marshalling Example (cont)



Marshalling Vs Serialization

- If you are using pytho[illegible]shalling and serialization are considered to the sa[illegible]g
- More complicated in Java
- Marshalling also records the codebases (location of object class definitions)
- Therefore, it treats remote objects differently and does more than serialization

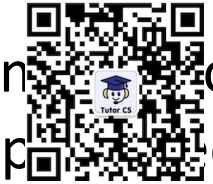
RPC Failures

- Like other components in distributed systems there are many types of failure and it is very difficult to determine the cause of a failure (debug)
- If failure of an RPC occurs any one of the following situations could have occurred:
  - The action was successfully performed by the remote server but the reply was lost
  - The server dies before starting the work
  - The request never reaches the server
  - The client dies after it sends the request but before it receives the response

RPC Success Modes

- This leads to different RPC success modes namely:
  - **Exactly once:** The RPC will execute once never more, never less. Expensive to implement but most like local calls
  - **At most once:** The client will only make one attempt to execute the RPC. If it works good, but if it fails the client will not attempt to repeat the RPC
  - **At least once:** The client will make multiple attempts to execute the RPC until it receives an acknowledgement even if the RPC executed on the remote host and acknowledgement of this was lost
  - **Idempotent:** The RPC can be repeated without a change

XML-RPC



- This is an RPC implen on which uses XML to encode its calls and as its transp chanism
- Calls can have multiple parameters and one result
- Parameters can be one of a few data types but these data types can be complex (For example, an array of integers would be a complex data type)
- Can be used in multiple languages C++, python, php and Java

JSON-RPC

- Very similar to XML-RPC but it uses JSON rather XML
- It also allows for notifications which are calls to the server which do not require a response
- It also allow for multiple calls to be sent to the server which can be answered out of order

XML vs JSON

- XML is more verbose than JSON
- XML is more difficult to parse
- JSON represents data as a map rather than a tree
- This has lead to criticism of XML-RPC in the 2010s

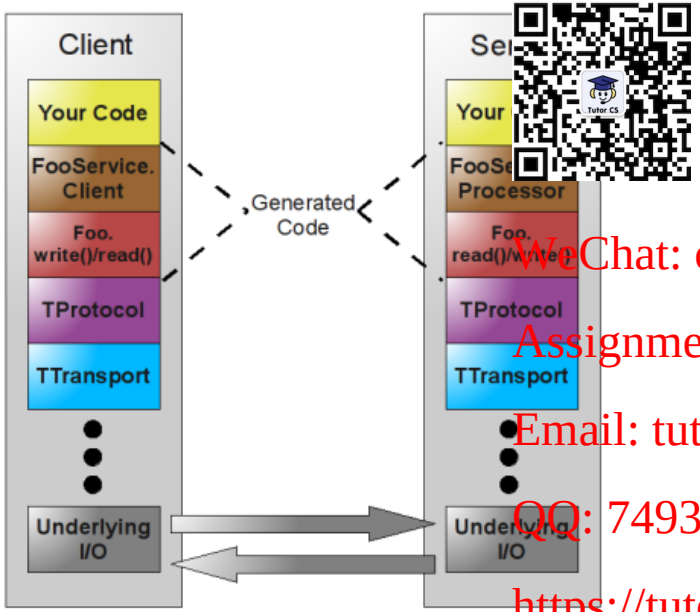
Modern RPC

- Apache Thrift
- gRPC

Apache Thrift

- Created by Facebook
- Now an Apache Project
- Simple Interface Definition Language
- Efficient Serialization in Space and Time - Variable Protocols
- Support for different Languages
- Code Generators for Glue Code
- Soft Versioning to allow interface and data type evolution between teams

Apache Thrift



From:; A.Prunicki, Thrift Overview, http://jnb.ociweb.com/jnb/jnbJun2009.html

gRPC

- Developed at Goo 2015
- Uses Google Protocol Buffers as the interface description language
- Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data
- Similar to XML but smaller faster and simpler

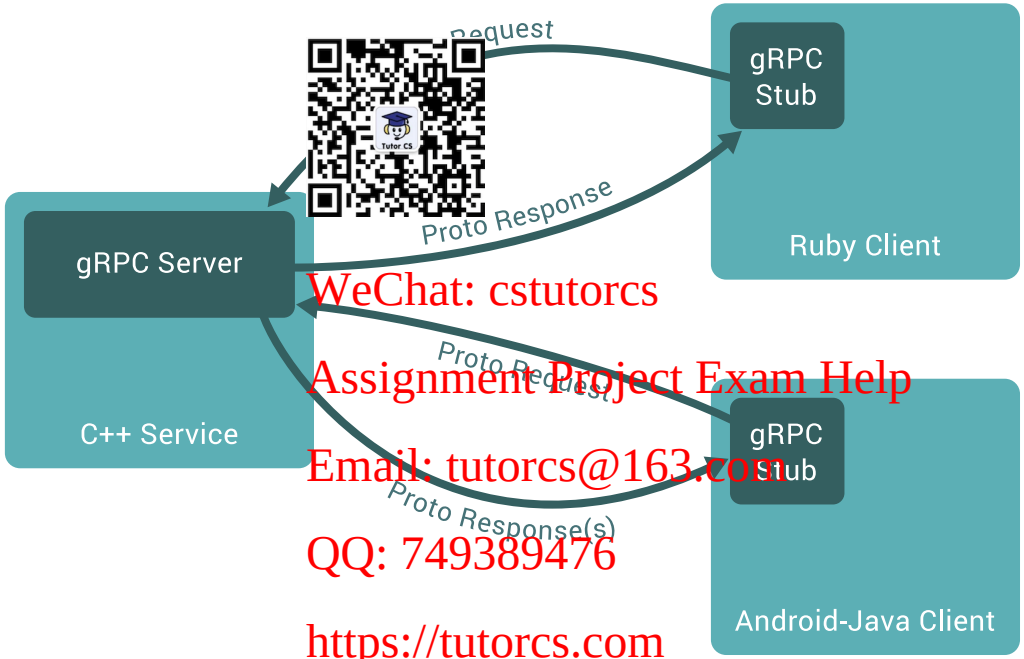
Google Protocol Buffers


```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}
```

Google Protocol Buffers


```
Person person;
person.set_name("John Doe");
person.set_id(1234);
person.set_email("jdoe@example.com");
fstream output("myfile", ios::out | ios::binary);
person.SerializeToOstream(&output);
```

gRPC



https://grpc.io/docs/guides/

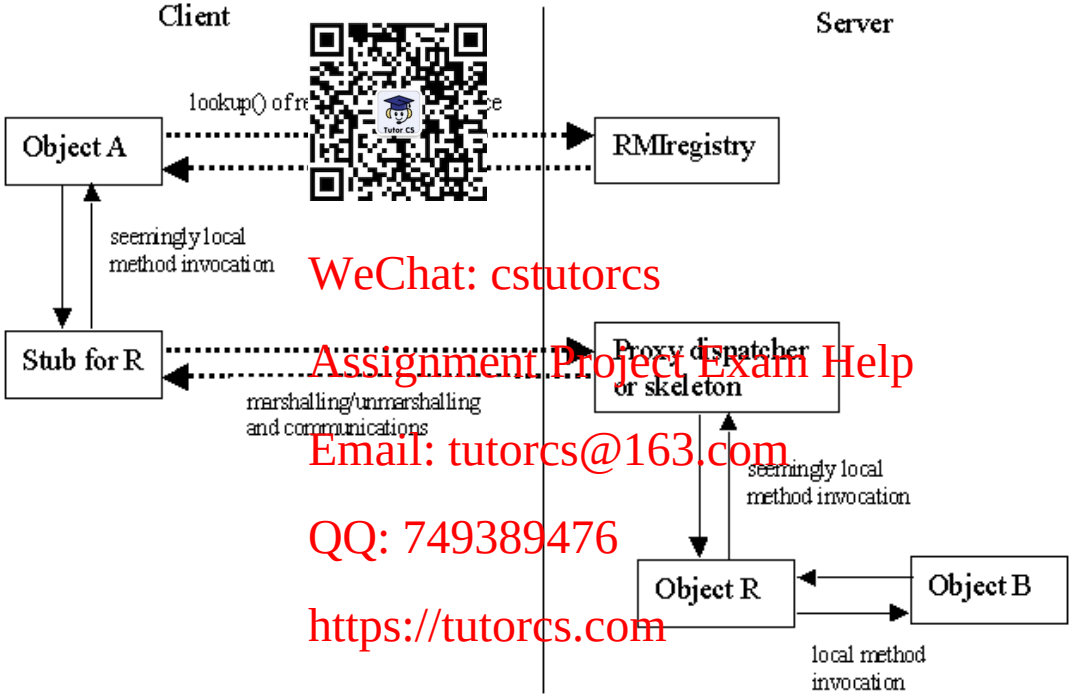
Remote Method Invocation (RMI)

- RPC does not provide support for object abstraction
- Java's RMI includes support for the direct transfer of serialized classes
- It also includes support for distributed garbage collection
- To achieve this a class must implement the Remote or UnicastRemote interface to make them Remote Objects

Java Remote Objects

- Remote objects are the exact same as local objects in Java
- References are used to identify objects in Java
- Java applications will never possess the reference to the remote object
- A proxy object known as a stub is used to represent this object locally and the stub is responsible for marshalling of messages and their delivery in a similar fashion to RPC

Remote Method Invocation (RMI)

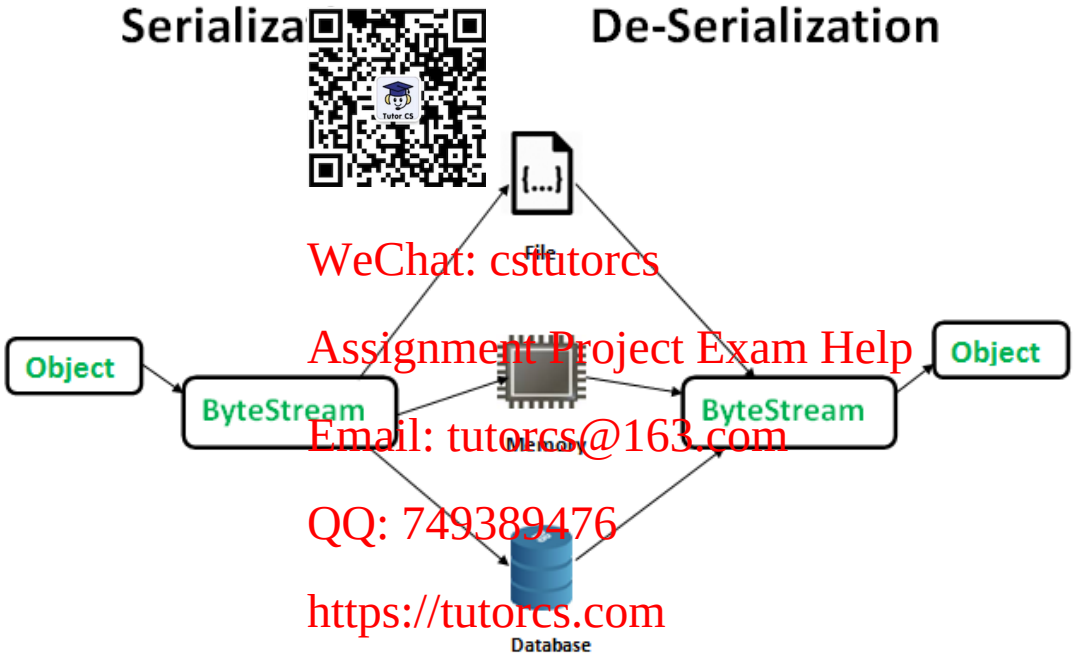


https://cseweb.ucsd.edu/classes/sp16/cse291-e/applications/ln/lecture3.html

RMI Passing By Reference

- In Java all values are by reference rather than by value
- This is problematic with RMI
- To determine which objects can be passed by RMI Java uses the simple rule of only allowing objects which implement Remote to be passed
- To transfer objects Java uses the process of serialization

Java Serialization



https://www.geeksforgeeks.org/serialization-in-java/

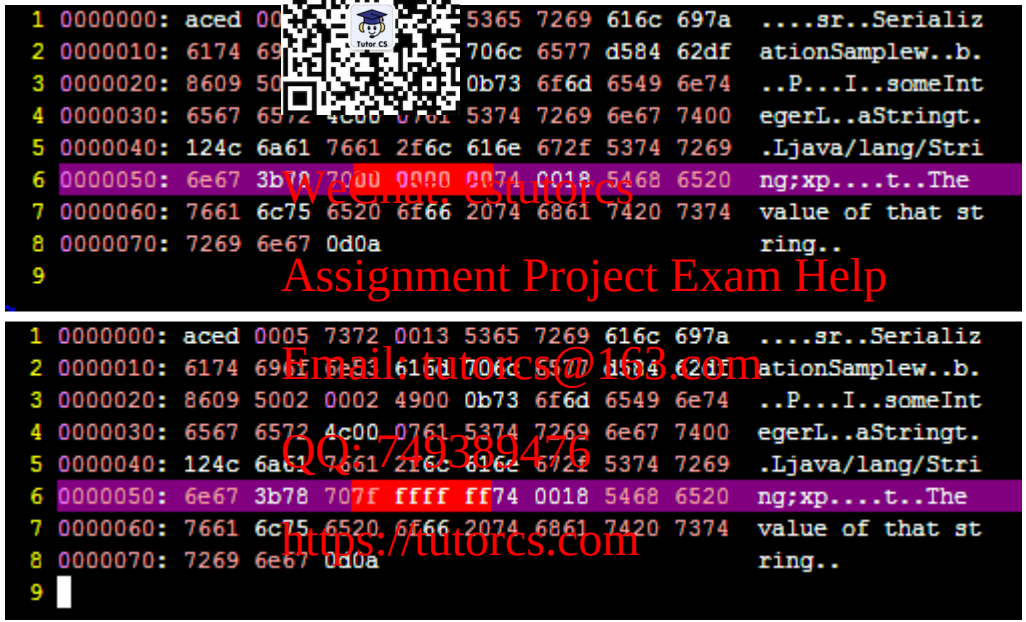
Java Serialization

- Serialization is the conversion of the object to a Byte Stream which contains information on the class, the member variables, the type of the member variables and the values of this particular instance
- This allows the class to be recreated after it is transferred across the network
- We can use the following code to examine a Java Byte Stream

Java Serialization Example


```
import java.io.*;
import java.util.*;

public class SerializationSample implements Serializable {

   private String aString = "The value of that string";
   private int    someInteger = 0;

   private transient List<File> unInterestingLongLongList;

   public static void main( String [] args ) throws IOException  {

      SerializationSample instance = new SerializationSample();

      ObjectOutputStream oos = new ObjectOutputStream(

                     new FileOutputStream(new File("o.ser")));

      oos.writeObject( instance );
      oos.close();
   }
}
```

Java Serialization Example



Java RMI Remote Interface Example

```
package com.mkyong.rmiint
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface RMIInterface extends Remote {
    public String helloTo(String name) throws RemoteException;
}
```

Java RMI Server Example


```
package com.mkyong.rmiserver;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import com.mkyong.rmiinterface.RMIInterface;
public class ServerOperation extends UnicastRemoteObject implements RMIInterface{
   private static final long serialVersionUID = 1L;
   protected ServerOperation() throws RemoteException {
      super();
   }
   @Override
   public String helloTo(String name) throws RemoteException{
      System.err.println(name + " is trying to contact!");
      return "Server says hello to " + name;
   }
   public static void main(String[] args){
      try {
         Naming.rebind("//localhost/MyServer", new ServerOperation());
         System.err.println("Server ready");

      } catch (Exception e) {
         System.err.println("Server exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Java RMI Client Example


```
package com.mkyong.rmiclient;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import javax.swing.JOptionPane;
import com.mkyong.rmiinterface.RMIInterface;
public class ClientOperation {
        private static RMIInterface look_up;
        public static void main(String[] args)
                throws MalformedURLException, RemoteException, NotBoundException {
                look_up = (RMIInterface) Naming.lookup("//localhost/MyServer");
                String txt = JOptionPane.showInputDialog("What is your name?");
                String response = look_up.helloTo(txt);
                JOptionPane.showMessageDialog(null, response);

        }
}
```

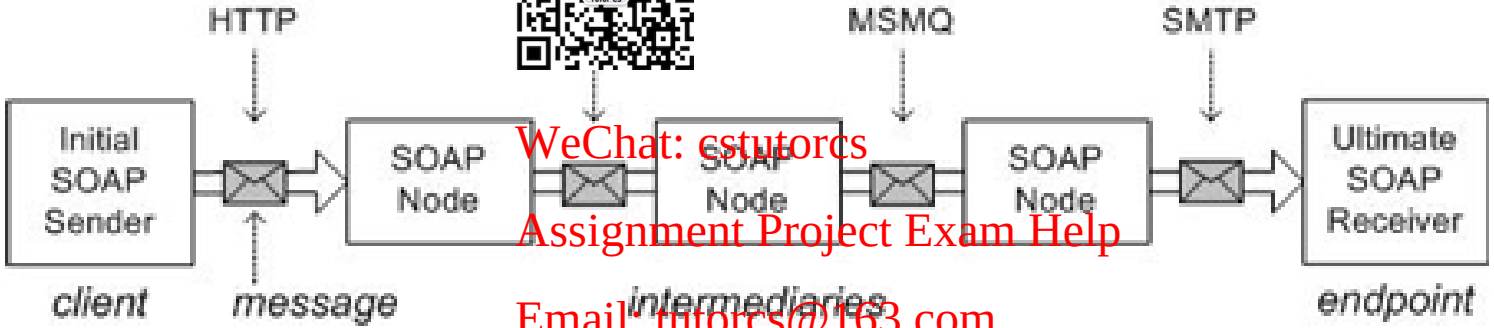
Simple Open Access Protocol (SOAP)

- Initially designed as a [illegible]ct Access protocol in 1998
- Designers were also i[illegible]d in the specification for XML-RPC
- The goal was to create a light weight messaging format that works with any operating system, programming language and platform
- It also aimed to allow access of remote objects using non HTTP traffic through firewalls

SOAP Characteristics

- The three main characteristics of SOAP are:
  - Extensibility (Other extensions such as security and addressing can be added to the original specification
  - Neutrality (SOAP can operate using any protocol such as HTTP, SMTP, TCP, UDP, or JMS)
  - Independence (SOAP can use any programming model)

SOAP Processing Model Example



SOAP Advantages and Disadvantages

Advantages

- SOAP's neutrality characteristic makes it suitable for use with any transport protocol.
- SOAP, when combined with HTTP post/response exchanges, tunnels easily through existing firewalls and proxies

Disadvantages

- When relying on HTTP as a transport protocol and not using Web Services Addressing or an Enterprise Service Bus, the roles of the interacting parties are fixed. Only one party (the client) can use the services of the other.
- The verbosity of the protocol and slow parsing speed of XML make it quite slow

Modern SOAP Usage

- Use of SOAP is in decline as other alternatives (such as REST which we will discuss next week) have better performance
- There are some cases, however, where the usage of SOAP is preferred.
- SOAP can be used to achieve:
  - more robust security through the WS-Security extension
  - ACID-compliant transactions with WS-Transaction and WS-Coordination

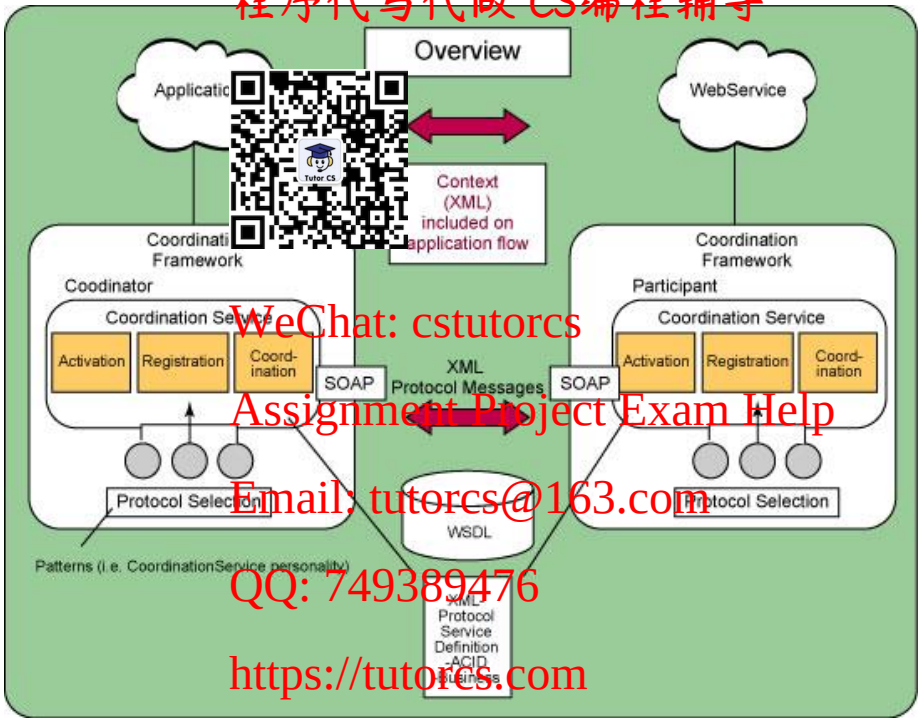
WS-Security

- WS-Security is an extensi[illegible]AP which provides “end-to-end” security
- HTTPS and TLS can be us[illegible]hieve a secure web connection
- These protocols, however, do not function correctly when using an application layer proxy server as the server would need to examine the request for routing
- This can be accounted for by storing the client key and certificate on the proxy server
- This is point-point security
- By using XML Signature and XML Encryption WS-Security can prevent this

ACID Compliant Transactions with SOAP

- If a set of operations on a collection of Web services that requires a mutually agreed outcome, WS-Transaction and WS-Coordination extensions can be used to achieve this
- WS-Transactions provides a series of protocols which allow activities to exhibit atomic behavior (It either succeeds or fails. It does not partially execute)
- WS-Coordination provides a definition of the behavior requirements and the operations supported for completion processing so that it is possible to determine if an activity succeeds or fails

Overview of WS-Coordination



https://www.ibm.com/developerworks/library/ws-wstx1/#figure3