GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

B.Sc. Software Development – Distributed Systems (2013)

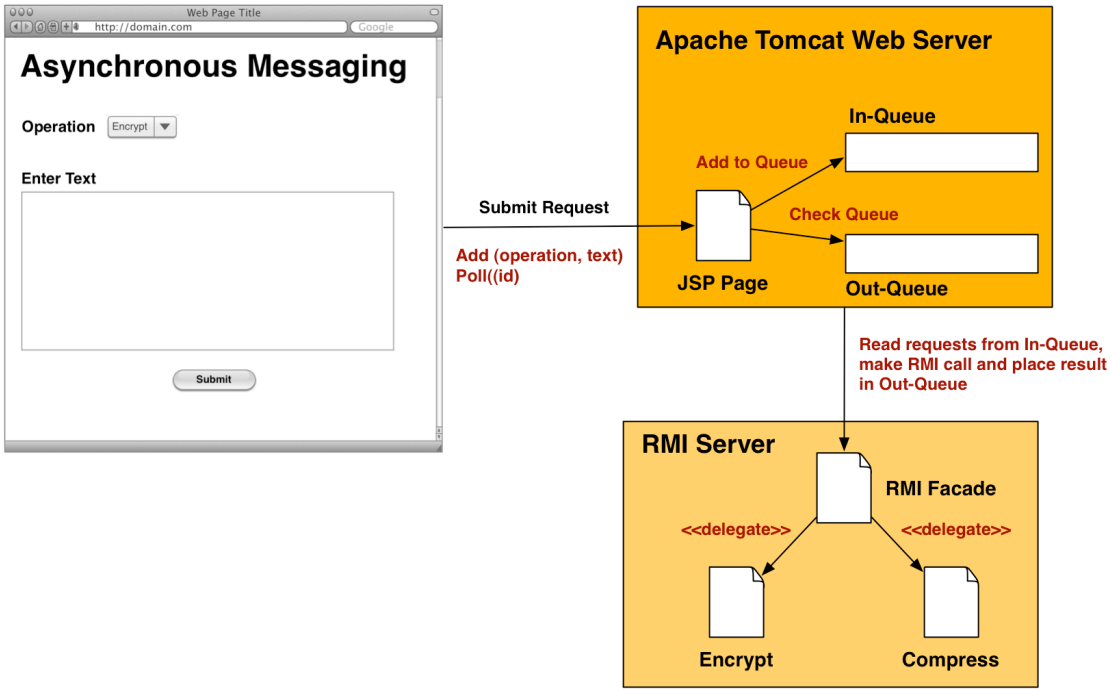
ASSIGNMENT DESCRIPTION & SCHEDULE

An Asynchronous RMI Encryption Service

Note: This assignment will constitute 30% of the total marks for this module.

1. Overview

You are required to use the Servlet/JSP and Java RMI frameworks to develop a remote, asynchronous encryption and compression service. A JSP page should provide users with the ability to compress or encrypt a file by selecting the operation they want and then entering some text. The HTML form information should be dispatched to a JSP page that adds the client request to an in-queue and then returns a unique ID to the client. The web client should poll the web server periodically and query if the request has been processed. Client request in the in-queue should be periodically removed and processed. The processing of a client request will require a RMI method invocation for a compression or encryption operation on a remote object on a different JVM. Once a response has been return by the remote object, it should be added to the out-queue on the Tomcat server and returned to the original web client when they next poll the server. The overall process is shown in the diagram below:



The rationale behind this assignment is to give you some "hands-on" experience programming an asynchronous remote communication system (a message façade). Asynchronous communication is an important topic in distributed computing, as it provides a degree of scalability if the number of potential requests is unknown or may vary wildly.

2. Minimum Requirements

1. Use the package name ***ie.gmit*** **throughout** your application.
2. **RMI Service:** Create and implement a single remote interface using a façade pattern to aggregate the functionality of compression and encryption. The RMI service will be called by an object running inside Tomcat's JVM.
3. **Tomcat Application:** Use JSP pages or servlets to process client requests to either execute an operation on the RMI service or to check if a previous request has been processed. Service requests should be placed in an in-queue and completed requests added to an out-queue (note that the out-queue is only logically a queue – you might find other data structures more efficient…).

The stubs for the Tomcat application have been provided on Moodle. Please do not waste your time adding styling "bells and whistles" to the client. Additional marks will only be given for programming extras. An example of AES encryption in Java is given at the end of this document. Java has built-in support for file compression – see the java.util.zip package – you should only need to use the *GZIPOutputStream* and *GZIPInputStream* for this assignment.

3. Deployment and Delivery

- ***The project must be submitted by midnight on Friday 10th January 2014.*** The project must be submitted as a Zip archive ***(not a rar, 7-zip or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the heading *"An Asynchronous RMI Encryption Service (30%): Assignment Upload"* in the "Notices and Assignments" section of Moodle.
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Artifact	Category
ds.war	Tomcat web application archive containing servlet and queuing system. Should automatically deploy when placed in TOMCAT_HOME\webapps
ds.jar	Java library with RMI service. Should be executable by executing the following command from a console: java –cp ./crypto.jar ie.gmit.AsyncService
src	A directory that contains the packaged source code for your application.
README.txt	A text file outlining the main features of your application, with a set of instructions for running the programme.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Category	%Total

Minimum Requirements†	60%
Object Oriented Design	10%
Server-Side Enhancements	10%
Client-Side Enhancements	10%
Other	10%

†The mimimum requirements cover deployment and delivery of the project. Any deviation from the specifications outlined above will result the deduction of marks under the category.

Each of the categories above will be scored using the following criteria:

- 0–30%: Not delivering on basic expectation
- 40-50%: Meeting basic expectation
- 60–70%: Tending to exceed expectation
- 80-90%: Exceeding expectations
- 90-100%: Exemplary

Using the Java Crypto API

```
import javax.crypto.*;
import java.security.Key;

public class Snowden{
  public static void main(String[] args) throws Exception{
        String plainText = "The quick brown fox jumped over the lazy dogs.";

        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        Key key = keyGen.generateKey();

        /* Advanced Encryption Standard (AES)
         * A newer encryption standard that supersedes the Data Encryption Standard (DES).
         * In worldwide use, including the US government. Allows for key sizes of 128, 192,
         * or 256 bits. DES encryption has a maximum key size of 56 bits, rendering it too
         * weak for use in modern security environments. None of this really matters anyway,
         * as the NSA can probably break AES of any key strength...
         */
        Cipher cypher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        byte[] bytes = plainText.getBytes("UTF8");

        //Encrypt
        cypher.init(Cipher.ENCRYPT_MODE, key);
        byte[] cypherText = cypher.doFinal(plainText.getBytes());
        System.out.println(cypherText);

        //Decrypt
        cypher.init(Cipher.DECRYPT_MODE, key);
        byte[] originalText = cypher.doFinal(cypherText);
        System.out.println(new String(originalText));
  }
}
```