SUDAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
COLLEGE OF GRADUATE STUDIES

Investigating the Performance of ASYCUDA System Based on RMI and REST Technology

التحقق من آداء نظام الاسيكودا بناءاً على تقنيتي "RMI" و "REST"

A Thesis Submitted in Partial Fulfillment of the Requirements of Master Degree in Computer Science

By:
Mohammed Ashraf Ibrahim Ahmed

Supervised By:
Dr. Nisreen Beshir Osman

2019

Sudan University of Science & Technology

College of Graduate Studies

جامعة السودان للعلوم والتكنولوجيا

كلية الدراسات العليا

كلية الدراسات العليا

Ref: SUST/ CGS/A11

Approval Page

(To be completed after the college council approval)

Name of Candidate: Mohammed Ashraf Ibrahim Abd Alkarim

Thesis title: Investigating the performance of ASyCUDA System Based on RMI and REST Technologies

التحقق من اداء نظام ASyCUDA الذي تم بناءه بتقنيتي "RMI" و "REST"

Degree Examined for: MSc in Computer Science

Approved by:

1. External Examiner

Name: Ali Ahmed

Signature: AWAL Date: 13/12/2018

2. Internal Examiner

Name: Wafaa Faisal Mukhtar

Signature: Date: 13/12/2018

3. Supervisor

Name: Nisreen Beshir

Signature: Date: 13/12/2018


بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ
اقْرَأْ بِاسْمِ رَبِّكَ الَّذِي خَلَقَ ۝١ خَلَقَ الْإِنسَانَ مِنْ عَلَقٍ ۝٢ اقْرَأْ
وَرَبُّكَ الْأَكْرَمُ ۝٣ الَّذِي عَلَّمَ بِالْقَلَمِ ۝٤ عَلَّمَ الْإِنسَانَ
مَا لَمْ يَعْلَمْ ۝٥
سُورَةُ الْعَلَقِ

Dedication

to my dear mother my life,

to spirit of my father,

who carried me on his shoulders

to the first day of school in my life

to spirit of my sister Reem

resistance icon and patience

to my teachers ,my sons and my wife

thanks Allah cause you are in my life

Acknowledgment

I extend my heartfelt thanks and gratitude to the University of Sudan for giving me this great opportunity to be among its students and to have the opportunity to receive a great deal of knowledge.

I also thank all the teachers who taught during this period and added to me a lot of experience and knowledge maturity that appeared in my career and how my scientific and practical abilities evolved.

I especially thank my supervisor for this research, who took my hand and helped me correct my various mistakes. Finally, I thank the Sudanese customs authority, the institution I am currently working on, which has given me the necessary administrative help.

Abstract

Sudan Customs uses “ASYCUDA” [2] software for customs clearance, it follows the architecture of n-tier, the application is split into three separate units in a make server and the user's PC share their messages through the web by using RMI technology, this leading the investigate on how can we enhance the current Sudan Customs System “ASYCUDA” performance through reduce it’s time response by using a new framework different to RMI. Related works shows that a REST Json based web service could be the best alternative choice for RMI replacement, it indicates that REST Json-base web service are more faster because of that, this research investigate exactly on time response comparison between RMI which represents the current framework that ASYCUDA based on it, two experimental models were designed, the first model represents the use of RMI technology and the second model represents the use of REST JSON based web service technology. Models consist of two parts, server application and client application. Results of those two models showed that RMI still better than REST in time response. We searched again for scientific justification; we explained it at the end of this research.

المستخلص

الجمارك السودانية تستخدم تطبيق الاسيكودا للتخليص الجمركي، و الذي يتبع معمارية الطبقات المتعددة، التطبيق ينقسم الي ثلاثة وحدات مستقلة تسمح للمخدم و اجهزة المستخدمين بتبادل البيانات من خلال تقنية ال .RMIهذا ما قادنا الي التحقق في كيفية تحسين آداء نظام الاسيكودا الحالي من خلال تقليل زمن الاستجابة و ذلك من خلال استخدام منظومة تقنية بديلة لل .RMIأظهرت الدراسات السابقة أن ال REST web service التي تستخدم كائن ال Json تعد هي البديل الافضل ، حيث اظهرت المقارنات ان تلك الاخيرة اسرع من ناحية زمن الاستجابة من تلك المستخدمة في نظام الاسيكودا. في هذا البحث تم تصميم نموزجين عمليين الاول نموزج يحاكي عمل تقنية ال RMI و الاخر يحاكي عمل تقنية ال REST Json based web service. كلا النموزجين يتكونان من تطبيق مخدم و تطبيق تابع . على عكس المتوقع اظهرت النتائج العملية ان تقنية ال RMI تعطي نتائجا افضل من ناحية سرعة الاستجابة ، قمنا بالبحث مرة اخري عن التفسير العملي لهذه النتائج و شرحها في نهاية البحث.

Table of the Contents

Chapter One

1. Introduction

1.1. Overview

According to ISO/IEC 25010 [1], Time-behavior beside resource utilization and capacity are the sub-characteristics of the performance efficiency characteristic. Response time is the key and the most important factor in time-behavior.

Time is an important factor in the process of customs clearance. The economy of countries depends largely on the movement of export and import of exported goods, and this requires customs clearance procedures. Traders are losing huge sums because of the congestion of goods at the port. There are goods that do not bear delays in customs clearance. Because of that this research focuses only on the time response.

1.2. Research Problem

1. There is a need to enhance the performance of the ASYCUDA.
2. There is no study that investigated the performance enhancement of the ASYCUDA.

1.3. Research importance

The importance of the research was related to improving the quality of the current ASYCUDA system, because the time factor is very important in reducing the time of clearance and satisfaction of clearance partners like Customs authority, clearance agents, importers, exporters and Taxation chamber, and Because acceleration of customs clearance may be a reason for increasing the commercial activity of exported and imported goods, which leads to the increase of customs and tax revenues, which may lead to a reduction in the rates of duties imposed on goods because the realization of the estimated financial ceiling annually and imposed by the Ministry of Finance relative availability. Briefly the importance is improving the quality and its impact on the system and the organization.

This research will help identify techniques proactively, before risking starting the process of reengineering an existing system

1.4. Research Objectives

- Design two experimental models; one of them uses RMI and the other uses REST Json base web service.
- Perform experiments and analyze results.
- Compare the two technologies (RMI and REST) according to these experimental results.

1.5. Structure of the thesis

The first chapter introduces this thesis by explaining what is the “Response Time” and its importance in this research. And then we have explained the problem field of investigation and its importance.

The second chapter gives theoretical background on the investigated technologies, namely “RMI” and “REST Json Based Web Service”, explain their components, and how they works. This chapter also explains what the ASYCUDA system is, and the technical framework it uses and its relationship to the mentioned technologies. Related works presents at the end of this chapter and how they were used in this investigation.

Chapter three concerned with the methodology by explaining the two experimental models, RMI and REST Json based models and how they were designed, there is a comprehensive documentation of these models. And then how experiments were performed and measurements recorded.

Chapter 4 means the results of the measurements recorded from the experiments conducted in the previous chapter. It also includes analyzing the results of these measurements, investigating their causes and searching for scientific justifications.

The final outcome of this investigation, in addition to the recommendations, is presented In Chapter 5.

Chapter 2

2. Literature Review

2.1. Remote Method Invocation (RMI)

RMI is part of the core Java API and has been enhanced for JDK 1.2 (Java 2 platform) in recognition of the critical need for support for distributed objects in distributed application development [3].

With RMI, you can get a reference to an object that "lives" in a remote process on remote hosts and invoke methods on it as if it were a local object running within the same Java virtual machine as your code. Each remote object implements a remote interface that specifies which of its methods can be invoked by clients. All object interfaces are written in Java since RMI is a Java-only distributed object scheme. Java RMI provides the following elements:
Remote objects implementations.
Client interfaces, or stubs, to the remote object.

1. A remote object registry for finding objects on the network.
2. A network protocol for communication between remote objects and their client (this protocol is the JRMP, i.e. Java Remote Method Protocol).
3. A facility for automatically (activating) remote objects on demand.

Prior to RMI, a distributed application involved socket programming, where a raw communication channel was used to pass messages and data between two remote processes. The programmer needed to define a low-level message protocol and data transmission format between processes in the distributed application. With RMI, you can "export" an object as a remote object, so that other remote processes/agents can access it 566 directly as a Java object. RMI handles all the underlying networking needed to make the remote method calls work.

There are three layers that comprise the basic RMI architecture.

1. The stub-skeleton layer, which provides the interface that client and server application objects use to interact with each other.
2. The remote reference layer, which is the middleware between the stubkkeleton layer and the underlying transport protocol.
3. The transport protocol layer, which is the binary data protocol that sends remote object requests over the wire. The client uses the client-side stub to make a request of the remote object.

The stub forwards the method invocation request through the remote reference layer by marshaling the method arguments into serialized form and asking the remote reference layer to forward the method request and arguments to the appropriate remote object. The remote reference layer converts the client request into low-level RMI transport requests, i.e., into a single network-level request and sends it over the wire to the remote object. On the server, the server-side remote reference layer receives the transport-level request and converts it into a request for the server skeleton that matches the referenced object. The skeleton converts the remote request into the appropriate method call on the actual server object.

This involves un-marshaling the method arguments into the server environment and passing them to the server object. Arguments sent as remote references are converted into local stubs on the server, and arguments sent as serialized objects are converted into local copies of the originals. If the method calls generates a return value or an exception, the skeleton marshals the object for transport back to the client and forwards it through the server reference layer.

RMI provides some basic object services on top of its remote object architecture that can be used by the distributed application designer. These services are:

1. Naming registry Service. A server process needs to register one (or more) RMI-enabled objects with its local RMI registry using a name that clients can use to reference it. A client can obtain a stub reference to the remote object by asking for the object by name.
2. Distributed Garbage Collection. This is an automatic process that the application developer does not have to worry about.
3. Object Activation Service. This service is new to RMI as of version 1.2 of the Java 2 platform. It provides away for a server object to be activated automatically when a client requests it.
4. The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object.

When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM).
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM).

3. It waits for the result.
4. It reads (unmarshals) the return value or exception.
5. It finally, returns the value to the caller.

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.

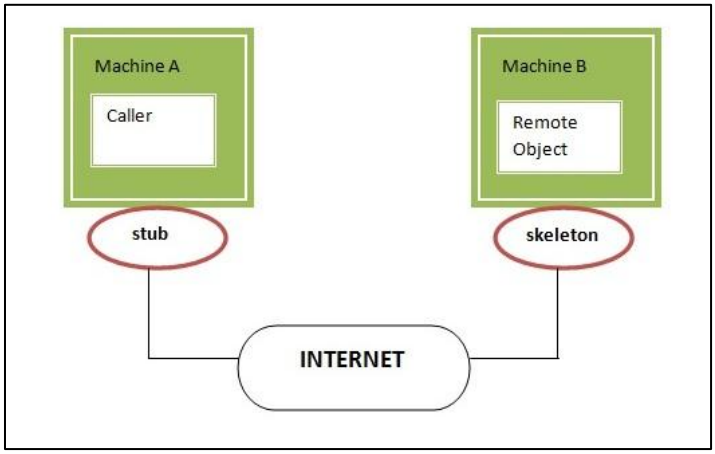


Figure 2.2: RMI structure.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.

In the RMI application, both client and server interact with the remote interface. The client application invokes methods on the proxy object; RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.

2.2. REST JSON Based Web Service

The term representational state transfer was introduced by Roy Fielding. REST style architecture is client server architecture in which client sends request to

server then server process the request and return responses. These request and responses build around the transfer of representations of resources. A resource is something that is identified by URI. Representation of resource is typically a document that captures the current or intended state of a resource. REST is less strongly typed than SOAP. The REST language is based on the use of nouns and verbs. REST does not require message format like envelope and header which is required in SOAP messages. So as XML parsing is also not required bandwidth requirement is less. Design principle of REST is as follows- addressability, statelessness and uniform interface. Addressability- REST models the datasets to operate on as resources where resources are marked with URI. A uniform and standard interface is used to access the rest resources i.e. using fixed set of HTTP methods. Every transaction is independent and unrelated to the previous transaction as all data required to process the request is contained in that request only, client session data is not maintained on server side therefore server responses are also independent.

These principles make the REST application simple and lightweight. The web application which follows the REST architecture we call it as RESTful web service. Restful web services uses GET, PUT, POST and DELETE http methods to retrieve, create, update and delete the resources [4].

2.3. RMI, SOAP, REST Comparison:

Refer to "Figure 2.1", by putting these three system's communication technologies (RMI, SOAP, REST) in balance; we found that the SOAP xml based web services were on average ~4.3 (4 for best case) times larger than RMI JRMP messages, on other hand Rest JSON base web service were also 5 to 6 (5 for worse case) times lesser than SOAP xml based web services, by putting these all in one balance we can say that Rest JSON base web service are ~ 1 time less than RMI .

But before going into this comparison the subject of this research, we must look deeply at these techniques, so that we understand the structure and how to work. Why did we compare the three technologies together and did not just a direct comparison between RMI which represent the current framework of ASYCUDA and the REST Json Based WS which represent the suggested recently the recommend replacement of the past one RMI? Simply because these two

technologies did not appear at the same time so we had to find a mediator compares the two of them which is SOAP technology.

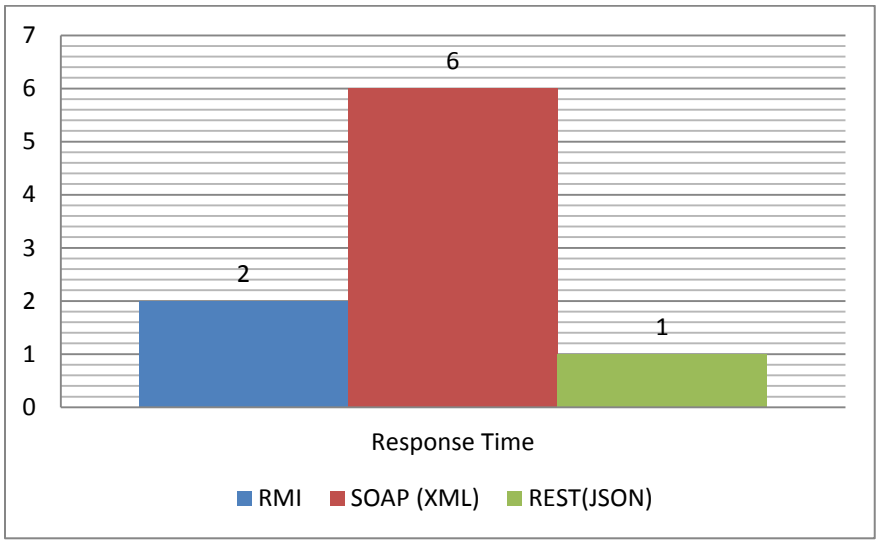


Figure 2.1: RMI, SOAP, REST Comparison

2.4. ASYCUDA Technical Overview

The ASYCUDA World systems are written entirely in Java; it is internet-based and uses Java web start to provide the latest Client applicants using your favorite web browser.

The ASYCUDA World general architecture is a state of the art n-tier system composed of modular products. Final user products are e-Document applications. It is based on SOClass™ framework which respects the four major aspect of system security as described overleaf.

Current edition of ASYWorld developed using SO-Class Framework, designed to offer solutions based on an 3-tier application model, in which user interface application resides on the end-users' computers, business logic resides on a centralized computer, and data requirements are handled by another computer managing a database.

2.5. SOClass™ Framework

SOClass framework is designed by Strategy Object Company to offer solutions based on an n-tier model. An n-tier application program is distributed

among three or more separate computers or logical layers in a distributed network environment. The most common form of n-tier (meaning ‘some number of tiers’) is the 3-tier application, in which user interface application resides on the end-users’ computers, business logic resides on a centralized computer, and data requirements are handled by another computer managing a database [5].

In addition to their induced orderliness of programming, n-tier applications have the obvious advantage that any of the tiers can run on a most appropriate processor or operating system, offering great scalability and capacity of evolution.

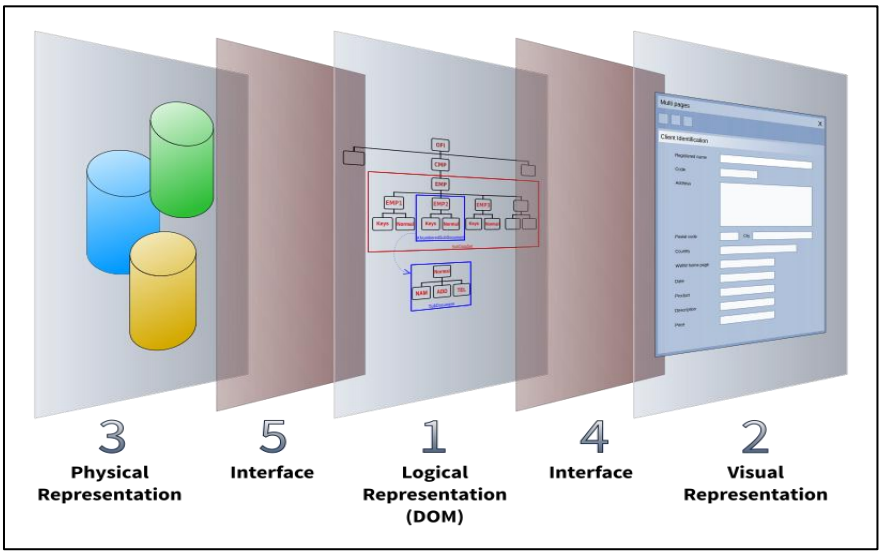


Figure 2.3: SOClass framework distributed systems.

As SOClass is an entirely Java system, it faithfully observes the popular “run anywhere” concept, as popularized by Java vendors. The distinct advantage for customers, and in particular governments, being that SOClass implementations can comply with IT development policies, previously established. For example, SOClass can work with any type of database management system supporting a JDBC driver or an ODBC driver trough a JDBC-ODBC bridge.

As a data-centric object, the e-document revolves around the data and logic it embeds. The SOClass data model is an hierarchical representation of the data contained in the e-document. Data can be viewed as a tree made of branches and data leaves – or data elements. This hierarchical representation proves natural and facilitates the analysis of the functional requirements. In addition, it eases the coding of the business logic through rules attached as adequate to tree nodes or leaves.

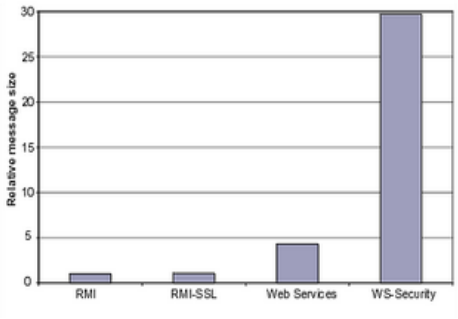
The Graphic User Interface (GUI) tier can be specific to each application, and each document, or harmonized around SOClass standard design. SOClass visual tools and examples allows beginners to quickly develop great document visual interfaces – or document skins – but also render possible for advanced programmers to plug-in more complex proprietary ones. The standard SOClass visual representation of a document comprises one or more forms, each comprising one or more pages. Finally the main thing here in this part is that SOClass Framework based on RMI, Remote Method Invocation, provides client-server communication interface.

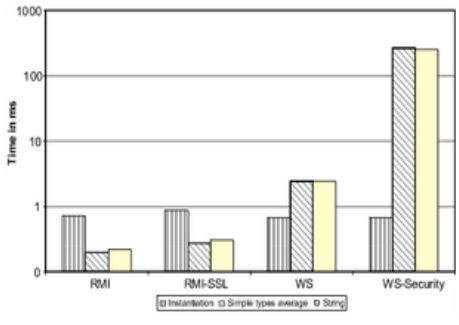
2.6. Related Work

Performance comparison of RMI had been done, (SOAP) Web services using a subset of the performance assessment framework. They measured the round trip method invocation times and the instantiation times, the round trip time expresses the overhead of remote method invocation [6].

Web services have been on average ~9 times slower than RMI. While the size of the RMI binary messages is related to the actual binary size of the data, the size of SOAP messages is related to the lexical length of data, data type and variable names (which are used for tag names and for xsi:type attributes). For creating and reading of SOAP messages Web services use XML serialization.

Binary serialization is an order of magnitude more efficient than XML serialization. Web services SOAP messages were on average ~4.3 times larger than RMI JRMP messages. When transferred over HTTP additional header information is added which further increases the message size.





This article analyses two most commonly used distributed models in Java (SOAP) Web services and RMI (Remote Method Invocation). The paper

contributes to the understanding of functional and performance related differences between SOAP WS and RMI.

Performance comparison on of SOAP based and RESTful web services based on different metric for mobile environment and multimedia conference is taken into consideration [7].

Evaluates the performance of both web services which provides the same functionalities in mobile computing environment. Two benchmarks are implemented based on float and string data type as parameter to the web service. The service client runs on mobile emulator. Results are captured for SOAP and RESTful web services in terms of total response time and message size. Message size in RESTful web services (in both cases) is 9 to 10 times lesser than size of SOAP based web services message. Similarly time required for processing and transmission is also 5 to 6 times lesser than SOAP based web services.

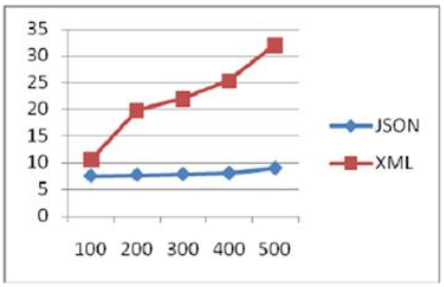
Lin, et al. compare the data transmission efficiency of JSON and XML, test environment was setup respectively, using these two data interchange format to transfer a same set of data from server-side, and gradually increase the amount of data to observe the changes of record delivery time, and thus indirectly to compare data transmission efficiency of them [8].

Number of array elements	Message Size (byte)				Time (Milliseconds)			
	SOAP/HTTP		*REST (HTTP)*		*SOAP/HTTP*		*REST (HTTP)*	
	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition
2	351	357	39	32	781	781	359	359
3	371	383	48	36	828	781	344	407
4	395	409	63	35	828	922	359	375
5	418	435	76	39	969	1016	360	359

Table 2.1: Performance Result Of SOAP and REST Web Services in Mobile Applicaton

From the test results, we can see JSON in the client's efficiency is much higher than the XML, and with the amount of data increases, the JSON's

deserialization time-consuming has no significant increase. However, XML with the amount of data increases, the client parse time appears to grow significantly.



Comparison chart of the test results for Client resolver efficiency

Automated performance testing is very important for large scale and distributed applications. Unsatisfactory performance may create functional and non-functional problems which must lead to inference in terms of time and resources [9].

As per ISO/IEC 9126 performance is measure in terms of efficiency meanwhile the parameters for measure are time behavior, resource utilization and efficiency compliance. In 2010 ISO/IEC FDIS 25010:2010(E) slandered efficiency changes to performance efficiency and the parameters are time behavior, resource utilization and capacity. Time behaviors refer to response time and throughput. Paper review some of the methodologies and tools used for software performance assessment and techniques used for gathering and capturing performance parameters (data), which are highly dynamic and uncertain.

Chapter 3

3. Methodology

We designed two experimental models. The first model represents the use of RMI technology and the second model represents the use of REST JSON based web service technology. Models consist of two parts, server application and client application.

Both models were designed using the Java programming language specifically JDK 1.7.0_80 and using NetBeans 8.1 as an IDE, server application run on Windows7 64 bit run on PC processor core i5,and 4 GB of RAM while client application run on PC with a dual core processor and OS Windows7 32 bit and 2 GB of RAM.

Of course, the two models were basically designed for time response comparison, so they work on same environment and same conditions. Client's applications are functionality similar to each other but they are different in way of server remotely calling, both server and client are in one LAN, so they connected together through intranet.

3.1. Models Design

The models idea is that a user interface represents clients which sends two literal strings to a server application, and then the server application returns a string to the user interface again**,** in these steps, we put two points to measure the system time response indicated by the sequence of operations as follows:

- **Point 1**: Time when a message was sent to the server application.
- **Point 2**: The time when the client gets the result from the server.

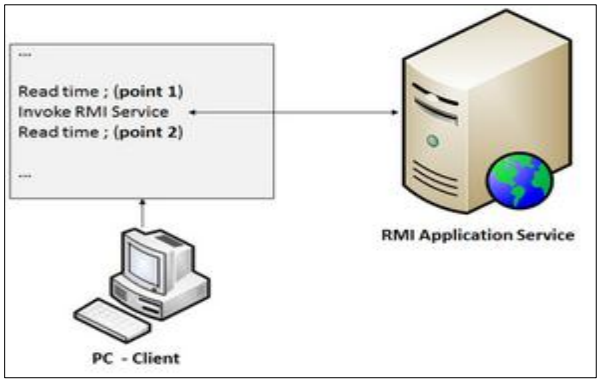


Figure 3.1: RMI Experimental Model

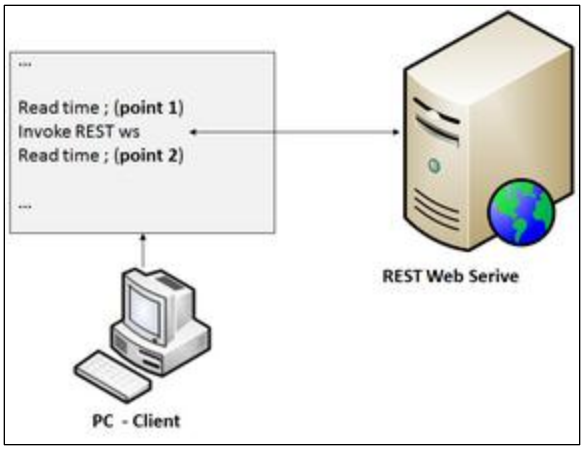


Figure 3.2: REST Experimental Model

We have implemented two models that simulate the same idea, but in two different ways, the first using the RMI and the other using REST JSON based.

We used the synchronization concept to emulate a certain number of users who are invoking a same function relies on server at the same time, we have achieved this by using multithreading calling ,of course the number of users has been taken into account in these experiments, finally we put our two types of clients in other PC before starting our experimental models.

3.1.1. RMI Model

This model is designed based on the RMI technique, which we explained early in the first chapter of this research. Depending on the needs of this technology in order to work we created three java ".jar" applications, the first one was "**RMIInterface**" application which represent the source of the remote interface, second one was "**RMIServer**" to provide the implementation of the remote interface and play the server role and the last application was "**RMIClient**" which represent the client where the object should invoked remotely. No additional java libraries needed, just the standard JDK1.7.

3.1.1.1 RMIInterface

This application includes just the remote interface called "**RMIInterface.java**", the project main source package "**rmiinterface**" contains a remote interface extend the java RMI Remote interface, is only one method named concat() and it declares RemoteException. Because the interface is a contract, we

designed it to be an independent application, and then built as a ".jar" project and then we included in the application's project library for both "**RMIServer**" and "**RMIClient**" applications.

By looking at "**RMIInterface.java"** (Figure 3.1) skeleton object has one method called "**great**()", method signature's parameters and it's returned value indicate that how the skeleton object could be invoked remotely and which type of value it will return.

```
1  package rmiinterface;
2
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  public interface RMIInterface extends Remote{
7  public String great(String clientID , String clientMessage ) throws
   RemoteException ;
8  }
```

Figure (3.1): RMIInterface.java

3.1.1.2 RMIServer

RMIServer source package contains RMIServer.java (Figure 3.2) class, which implement RMIInterface interface by extend the UnicastRemoteObject class. Because we extend the UnicastRemoteObject class, we must define a constructor that declares RemoteException. As you see at line 16, the server class override great() RMIInterface interface method, here is skeleton application logic, as you see this method return the result of String message contains "clientID" and "clientMessage" which were send as parameters. We added at line 19 a print message to determine the time when the server received this message from the client (Point 2). Line 25 will start the registry service on port "4444". Line 26 binds the remote object to the new name.

```
1  package rmiinterface;
2  import java.rmi.RemoteException;
3  import java.rmi.registry.LocateRegistry;
```

```
4  import java.rmi.registry.Registry;
5  import java.rmi.server.UnicastRemoteObject;
6  import java.text.SimpleDateFormat;
7  import java.util.Date;
8
9  public class server extends UnicastRemoteObject implements RMIInterface
   {
10
11   public server() throws RemoteException {
12   }
13
14   @Override
15   public String concat(String clientID, String clientMessage) throws
   RemoteException {
16       SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
   HH:mm:ss.SSS");
17  System.out.println(" Point 2 , Receipt " + clientMessage + " from " +
   clientID + "\t@:" + sdf.format(new Date()));
18       return clientID +" Server Replay your message was : " +
   clientMessage ;
19 }
20
21   public static void main(String[] s) throws RemoteException
22   {
23      Registry reg = LocateRegistry.createRegistry(4444) ;
24      reg.rebind("hi_server", new server());
25      System.out.println(" server is ready ... ");
26   }
27 }
```

Figure (3.2): RMIServer.java

3.1.1.3 RMIClient

This application consists of two classes and as we mentioned previously the "**RMIInterface.jar**" file. The first class is DoRMI Class and the other one is RMIClient class.

DoRMI (shown in Figure 3.3) Class represent one user or client invoke great() method on the server remotely. Each instance of this class defines a new instance of RMIInterface and it extends Thread class because we need of synchronization. As you see at line 27,28 we print when the client start calling the RMI method (Point 1), and when the client receipt the result from server (Point 3).

```
1
2  package rmiclient;
3
4  import java.rmi.RemoteException;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.logging.Level;
8  import java.util.logging.Logger;
9  import rmiinterface.RMIInterface;
10
11 public class DoRMI extends Thread {
12    static int serial ;
13    String clientName ;
14    RMIInterface ad ;
15
16    public DoRMI(String clientName) {
17       this.clientName = clientName;
18    }
19
20  @Override
21    public void run() {
22       String name = Thread.currentThread().getName();
23       SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
   HH:mm:ss.SSS");
24       try {
25          Thread.sleep(0);
26          synchronized (this) {
27 Date D1 = new Date() ;
28 System.out.println("<= Point 1,\t" + clientID + " start calling @ " +
   sdf.format(new Date()) + ", Duration "+ (new Date().getTime() -
   D1.getTime() ));
29       System.out.println("=> Point 3,\t" + ad.great(clientID, clientMessage)
```

```
    + "\t:@" + sdf.format(new Date()));
30              }
31          } catch (InterruptedException e) {
32              System.out.println(e.getMessage());
33          } catch (RemoteException ex) {
34              Logger.getLogger(DoRMI.class.getName()).log(Level.SEVERE,
    null, ex);
35          }
36
37      }
38  }
39
```

Figure (3.3): DoRMI.java

The RMIServer class (shown in Figure 3.4) lookup for RMIInterface hold the registry name “hi_server”, existed on a server with IP “localhost” on port “4444”.

```
1   package rmiclient;
2
3   import java.rmi.NotBoundException;
4   import java.rmi.RemoteException;
5   import java.rmi.registry.LocateRegistry;
6   import java.rmi.registry.Registry;
7   import java.text.SimpleDateFormat;
8   import java.util.Date;
9   import java.util.logging.Level;
10  import java.util.logging.Logger;
11  import rmiinterface.RMIInterface;
12
13  public class RMIClient {
14      public static void main(String[] args) {
15          Registry reg;
16          RMIInterface ad;
17          SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
    HH:mm:ss.SSS");
18
19          try {
```

```
20          reg = LocateRegistry.getRegistry("localhost", 4444);
21          ad = (RMIInterface) reg.lookup("hi_server");
22          for (int i = 0; i < 10; i++) {
23              DoRMI t = new DoRMI("client " + i);
24              t.ad = ad;
25              t.start();
26          } catch (RemoteException | NotBoundException ex) {
27              Logger.getLogger(RMIClient.class.getName()).log(Level.SEVERE,
    null, ex);
28          }
29      }
30  }
```

Figure (3.4): RMIServer.java

3.1.2. REST WS Model

This second experimental model consist of two applications, the RestWS as web service application and the RestWSClient as a user client interface, here we used additional external APIs needed to apply java web service technology which was JAX-RS 2.0.

We used apache-tomee-webprofile-7.0.1 as web server application. On the client side we used Jersey 2.5.1 (JAX-RS RI) .

3.1.2.1 RestWS

According to the Rest web service technology concept we put our “greet()” method as a resource over a service path, which receipt two parameters “clientID” and “clientMessage” from the service path URL, and return a server String message. (Fig 3.5) display the source code of “RestWS” web resource.

```
1  package logic;
2
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5  import javax.ws.rs.core.Context;
6  import javax.ws.rs.core.UriInfo;
7  import javax.ws.rs.Consumes;
8  import javax.ws.rs.PUT;
9  import javax.ws.rs.Path;
```

```
10 import javax.ws.rs.GET;
11 import javax.ws.rs.PathParam;
12 import javax.ws.rs.Produces;
13 import javax.ws.rs.core.MediaType;
14
15 @Path("ws")
16 public class WsResource {
17
18    @Context
19    private UriInfo context;
20
21    @GET
22    @Produces(MediaType.APPLICATION_JSON)
23    @Path("great/{clientID},{clientMessage}")
24    public String great( @PathParam("clientID") String clientID
   ,@PathParam("clientMessage") String clientMessage ) {
25        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
   HH:mm:ss.SSS");
26        System.out.println(" Point 2 , Receipt " + clientMessage + " from " +
   clientID + "\t@:" + sdf.format(new Date()));
27        return clientID +" Server Replay your message was : " +
   clientMessage ;
28    }
29
30    @PUT
31    @Consumes(MediaType.APPLICATION_JSON)
32    public void putJson(String content) {
33    }
34 }
```

Fig 3.5: RestWS resource

3.1.2.2 RestWSClient

Our Rest WS client application consist of two java classes , “DoRESTCall” the class that representatives 10 users calling the above RestWS at the same time by using Threads (see Fig:3.6) , and the project class that trigging that class (see Fig:3.7).

```
1 package restclient;
```

```
2
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5  import javax.ws.rs.client.Client;
6  import javax.ws.rs.client.WebTarget;
7
8  public class DoRESTCall extends Thread {
9
10    private WebTarget webTarget;
11    private Client client;
12    String clientID;
13    String clientMessage;
14
15    public DoRESTCall(WebTarget webTarget, Client client, String clientID,
   String clientMessage) {
16        this.client = client;
17        this.webTarget = webTarget;
18        this.clientID = clientID;
19        this.clientMessage = clientMessage;
20
21    }
22
23    public void close() {
24        client.close();
25    }
26
27    @Override
28    public void run() {
29        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
   HH:mm:ss.SSS");
30        try {
31            Thread.sleep(0);
32            synchronized (this) {
33   Date D1 = new Date() ;
34                System.out.println("<= Point 1,\t" + clientID + " start calling @ "
   + sdf.format(new Date()));
35                WebTarget resource =
   webTarget.path(java.text.MessageFormat.format("great/{0},{1}", new
   Object[]{clientID, clientMessage}));
```

```
36              String result = resource.request(javax.ws.rs.core.MediaType.APPLICATION_JSON).get(String.class);
37              System.out.println("=> Point 3,\t" +result + "\t:@" + sdf.format(new Date()) +  ", Duration "+ (new Date().getTime() - D1.getTime()  ));
38          }
39      } catch (InterruptedException e) {
40          System.out.println(e.getMessage());
41      }
42  }
43 }
```

Fig: 3.6: DoRESTCall.java

```
1  package restclient;
2
3  import javax.ws.rs.client.Client;
4  import javax.ws.rs.client.WebTarget;
5
6  public class RestWSClient {
7
8      public static void main(String[] args) {
9          String BASE_URI = "http://localhost:8080/RestWS/webresources";
10
11         for (int i = 0; i < 10 ; i++) {
12             Client client = javax.ws.rs.client.ClientBuilder.newClient();
13             WebTarget webTarget = client.target(BASE_URI).path("ws");
14             DoRESTCall c = new DoRESTCall(webTarget, client, "c" + i, "Greating NO. " + i);
15             c.start();
16         }
17     }
18 }
```

Fig: 3.7: RestWSClient main class

3.2. Conduct experiments

When we run RMIClient application, we get Point's times through printing on the application consol. These times represent two kind of points, lines start with

"=>" indicates times of type **Point 1**, and that start with "<=" indicate times of type **Point 3**. Any kind of the two types of points existed for the 10 number of instances or by other word transactions starting at the same time. The results show the normal non-ordering starting / finishing remote method invocation of each instance because of using thread's synchronizations.

3.3. Measurements and results

3.3.1. RMI

1st attempt Client Console

```
<= Point 1,  c1 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c4 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c3 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c2 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c7 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c6 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c0 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c9 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c8 start calling @ 2018-08-07 10:57:39.925
<= Point 1,  c5 start calling @ 2018-08-07 10:57:39.925
=> Point 3,  c0 Server Replay your message was : Greating NO. 0
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c7 Server Replay your message was : Greating NO. 7
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c8 Server Replay your message was : Greating NO. 8
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c1 Server Replay your message was : Greating NO. 1
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c2 Server Replay your message was : Greating NO. 2
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c4 Server Replay your message was : Greating NO. 4
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c9 Server Replay your message was : Greating NO. 9
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c3 Server Replay your message was : Greating NO. 3
       :@2018-08-07 10:57:40.346 Duration was :421
=> Point 3,  c6 Server Replay your message was : Greating NO. 6
       :@2018-08-07 10:57:40.346 Duration was :421
```

```
=> Point 3,  c5 Server Replay your message was : Greating NO. 5
        :@2018-08-07 10:57:40.346 Duration was :421
```

2nd attempt Client Console

```
<= Point 1,  c1 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c5 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c4 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c9 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c8 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c2 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c6 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c3 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c0 start calling @ 2018-08-07 10:58:15.498
<= Point 1,  c7 start calling @ 2018-08-07 10:58:15.498
=> Point 3,  c0 Server Replay your message was : Greating NO. 0
        :@2018-08-07 10:58:15.888 Duration was :390
=> Point 3,  c8 Server Replay your message was : Greating NO. 8
        :@2018-08-07 10:58:15.888 Duration was :390
=> Point 3,  c5 Server Replay your message was : Greating NO. 5
        :@2018-08-07 10:58:15.888 Duration was :390
=> Point 3,  c4 Server Replay your message was : Greating NO. 4
        :@2018-08-07 10:58:15.888 Duration was :390
=> Point 3,  c2 Server Replay your message was : Greating NO. 2
        :@2018-08-07 10:58:15.904 Duration was :406
=> Point 3,  c3 Server Replay your message was : Greating NO. 3
        :@2018-08-07 10:58:15.904 Duration was :406
=> Point 3,  c1 Server Replay your message was : Greating NO. 1
        :@2018-08-07 10:58:15.920 Duration was :422
=> Point 3,  c6 Server Replay your message was : Greating NO. 6
        :@2018-08-07 10:58:15.920 Duration was :422
=> Point 3,  c7 Server Replay your message was : Greating NO. 7
        :@2018-08-07 10:58:15.920 Duration was :422
=> Point 3,  c9 Server Replay your message was : Greating NO. 9
        :@2018-08-07 10:58:15.920 Duration was :422
```

3rd attempt Client Console

```
<= Point 1,  c0 start calling @ 2018-08-07 10:58:39.401
<= Point 1,  c9 start calling @ 2018-08-07 10:58:39.401
```

```
<= Point 1,   c1 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c6 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c3 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c4 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c2 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c7 start calling @ 2018-08-07 10:58:39.401
<= Point 1,   c8 start calling @ 2018-08-07 10:58:39.416
<= Point 1,   c5 start calling @ 2018-08-07 10:58:39.416
=> Point 3,   c2 Server Replay your message was : Greating NO. 2
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c0 Server Replay your message was : Greating NO. 0
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c7 Server Replay your message was : Greating NO. 7
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c6 Server Replay your message was : Greating NO. 6
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c5 Server Replay your message was : Greating NO. 5
        :@2018-08-07 10:58:39.806 Duration was :390
=> Point 3,   c1 Server Replay your message was : Greating NO. 1
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c8 Server Replay your message was : Greating NO. 8
        :@2018-08-07 10:58:39.806 Duration was :390
=> Point 3,   c4 Server Replay your message was : Greating NO. 4
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c9 Server Replay your message was : Greating NO. 9
        :@2018-08-07 10:58:39.806 Duration was :405
=> Point 3,   c3 Server Replay your message was : Greating NO. 3
        :@2018-08-07 10:58:39.806 Duration was :405
```

3.3.2. REST WS

1st attempt Client Console

```
<= Point 1,   C0 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C1 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C2 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C3 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C6 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C7 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C4 start calling @ 2018-08-07 11:11:31.055
<= Point 1,   C5 start calling @ 2018-08-07 11:11:31.055
```

```
<= Point 1,  C8 start calling @ 2018-08-07 11:11:31.055
<= Point 1,  C9 start calling @ 2018-08-07 11:11:31.055
=> Point 3,  C3 Server Replay your message was : Greating NO. 3
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C4 Server Replay your message was : Greating NO. 4
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C8 Server Replay your message was : Greating NO. 8
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C6 Server Replay your message was : Greating NO. 6
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C9 Server Replay your message was : Greating NO. 9
        :@2018-08-07 11:11:31.102 Duration was :47
       => Point 3,  C2 Server Replay your message was : Greating NO. 2
                    :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C7 Server Replay your message was : Greating NO. 7
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C5 Server Replay your message was : Greating NO. 5
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C1 Server Replay your message was : Greating NO. 1
        :@2018-08-07 11:11:31.102 Duration was :47
=> Point 3,  C0 Server Replay your message was : Greating NO. 0
        :@2018-08-07 11:11:31.243 Duration was :188
```

2nd attempt Client Console

```
<= Point 1,  C6 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C2 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C1 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C5 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C0 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C8 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C7 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C3 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C9 start calling @ 2018-08-07 11:12:54.378
<= Point 1,  C4 start calling @ 2018-08-07 11:12:54.378
=> Point 3,  C6 Server Replay your message was : Greating NO. 6
        :@2018-08-07 11:12:54.378 Duration was :0
=> Point 3,  C5 Server Replay your message was : Greating NO. 5
        :@2018-08-07 11:12:54.394 Duration was :16
```

```
=> Point 3,  C8 Server Replay your message was : Greating NO. 8
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C0 Server Replay your message was : Greating NO. 0
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C1 Server Replay your message was : Greating NO. 1
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C7 Server Replay your message was : Greating NO. 7
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C9 Server Replay your message was : Greating NO. 9
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C4 Server Replay your message was : Greating NO. 4
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C2 Server Replay your message was : Greating NO. 2
        :@2018-08-07 11:12:54.394 Duration was :16
=> Point 3,  C3 Server Replay your message was : Greating NO. 3
        :@2018-08-07 11:12:54.394 Duration was :16
```

3rd attempt Client Console

```
<= Point 1,  C1 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C3 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C2 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C4 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C0 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C7 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C8 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C9 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C6 start calling @ 2018-08-07 11:13:16.393
<= Point 1,  C5 start calling @ 2018-08-07 11:13:16.393
=> Point 3,  C1 Server Replay your message was : Greating NO. 1
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C3 Server Replay your message was : Greating NO. 3
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C4 Server Replay your message was : Greating NO. 4
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C0 Server Replay your message was : Greating NO. 0
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C7 Server Replay your message was : Greating NO. 7
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C5 Server Replay your message was : Greating NO. 5
```

```
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C2 Server Replay your message was : Greating NO. 2
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C6 Server Replay your message was : Greating NO. 6
        :@2018-08-07 11:13:16.393 Duration was :0
=> Point 3,  C9 Server Replay your message was : Greating NO. 9
        :@2018-08-07 11:13:16.408 Duration was :15
=> Point 3,  C8 Server Replay your message was : Greating NO. 8
        :@2018-08-07 11:13:16.408 Duration was :15
```

Chapter 4

4. Results and Discussions

4.1. Results

attempt	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	188	47	47	47	47	47	47	47	47	47
2	16	16	16	16	16	16	0	16	16	16
3	0	0	0	0	0	0	0	0	15	15
Average	102	31	31	31	31	31	23	31	39	39

Figure (4.1): RMI time response duration in Milliseconds

Figure (4.1) shows that RMI time response duration in milliseconds, the time durations of the three attempts for the 10 threads (clients), then we calculated the average time response duration for each client and then we calculated the total average, note that the first attempt (C0) took much more time than the following attempts because RMI transport layer opens direct sockets to remote object hosts and uses binary protocol for communication and cashing the remote object in the jvm registry and this step doesn't needed for the following attempts, the total average of time response for RMI model was ≈ **38** milliseconds.

attempt	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9
1	421	421	421	421	421	421	421	421	421	421
2	390	422	406	406	390	390	422	422	390	422
3	405	405	405	405	405	390	405	405	390	405
Average	405	416	410	410	405	400	416	416	400	416

Figure (4.2): REST WS time response duration in Milliseconds

Figure (4.2) shows that REST WS time response durations, the total average of time response for REST WS model was ≈ **409** milliseconds. These results show that the REST JSON base web service was ≈ **10 times** slower than RMI service.

Looking at the results recorded in the experimental model, it can be seen that the time spent in the RMI model is decreasing with repeated attempts, noting that the first attempt took an exceptional time because the JRMP protocol cashing the registry in client's jvm, in contrast to the REST WS model, the times are almost too close because the HTTP protocol starts the same steps of communication with the server in each web service consuming attempt.

4.2. Discussions

The main difference between RMI and Web services in performance is the payload, the message protocol that uses for exchange data between distributed java virtual machines. RMI uses binary protocol with binary data that makes use of the Java object serialization for calling and returning the data.

RMI transport layer usually opens direct sockets to remote object hosts and uses binary protocol for communication it is JRMP (Java Remote Method Protocol) which is a Java proprietary protocol and it uses the TCP/IP as the underlying protocol. When transferred over HTTP additional header information is added which further increases the message size.

Java directly supports distributing java objects from any java application through Remote Method Invocation (RMI).This distributed-objects package simplifies communication among Java applications on multiple machines [10].

REST WS uses HTTP over TCP/IP, "there are two different roles: server and client. In general, the client always initiates the conversation; the server replies. HTTP is text based; that is, messages are essentially bits of text, although the message body can also contain other media. Text usage makes it easy to monitor an HTTP exchange.

HTTP messages are made of a header and a body. The body can often remain empty; it contains data that you want to transmit over the network, in order to use it according to the instructions in the header. The header contains metadata, such as encoding information; but, in the case of a request, it also contains the important HTTP methods. In the REST style, you will find that header data is often more significant than the body [11].

Chapter 5

5. Conclusions and Recommendations

5.1. Conclusions

Is the ASYCUDA should be faster if we re-engineering it's architectural design from using RMI to be a REST JSON based web service ?, the answer is no.

Experimental model shows that RMI is faster than REST WS, because the size of the binary serialized object is smaller than the same object as an XML or JSON representation, and because of difference of way of intercommunication between RMI and REST WS.

For all I agree with the opinion that says the web service is not a substitute or replacement of RMI technology. Each of these two methods has an advantages and disadvantages; different scope of work depends on the functional and technical requirements like compatibility and integration. Performance is also not just a quick time response.

Information security also requires other arrangements and studies. The development of ASYCUDA is my main concern. I will continue to research and study in this context.

5.2. Recommendations

The need to investigate security enhancement still exist. Because performance and security go in opposite directions, that requires exploration in different sources of knowledge. May be different quality factor could be investigated in future.

References

[1] Product Quality ISO/IEC 25010, 2011.

[2] asycuda.org (2018). ASYCUDA Software Versions [Online]. Available: https://asycuda.org/software/#

[3] Sanjay P. Ahuja and Renato Quintao, "Performance Evaluation of Java RMI: A Distributed Object Architecture for Internet Based Applications," Department of Computer and Information Sciences, University of North Florida, Jacksonville, 2000.

[4] Snehal Mumbaikar and Puja Padiya, "Web Service Based On SOAP and REST Principls," Department of Computer Engineering, R. A. I. T., Ramrao Adik Institute of Technology, India , 2013.

[5] Strategy Object (2018). Java Technologies in SOClass [Online]. Available: https://www.strategyobject.com/technologies/java-technologies/

[6] Matjaz B. Juric, Ivan Rozman, Bostjan Brumen, Matjaz Colnaric, Marjan Hericko, "Comparison of performance of Web services, WS-Security,RMI, and RMI-SSL" , Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia, 2005.

[7] Smita Kumari, and Puja Padiya, "Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration," Department of Computer science and Engineering, National Institute of Technology, Rourkela, INDIA, 2015.

[8] Boci Lin , Yan Chen , Xu Chen and Yingying Yu, "Comparison between Json and XML in applications on AJAX", International Conference on Computer Science and Service System, Transportation Management College Dalian Maritime University Dalian, China , 2015.

[9] Muhammad Sadiq , Muhammad Shahid Iqbal , Amizah Malip , Wan Ainun Mior Othman, "A Survey of Most Common Referred Automated Performance Testing Tools , ARPN Journal of Science and Technology ISSN", Faculty of

Computing (RIU) Islamabad , School of Computer Science, Anhui University, Hefei, China , Institute of Mathematical Science, University of Malaya, 2015.

[10] Larry Brown and Marty Hall (2002), Creating Clients and Servers with Java Sockets [Online]. Available: http://www.informit.com/articles/article.aspx?p=26350

[11] Ludovico Fischer (2013), A Beginner's Guide to HTTP and REST [Online]. Available: https://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340