WEB SERVICE PERFORMANCE ON HETEROGENEOUS SYSTEMS

A performance comparison between J2EE and .NET web services on heterogeneous systems

Bachelor Degree Project in Informatics
30 ECTS
Spring term 2013

Mikel Urquia Cortabarria

Supervisor: Henrik Gustavsson
Examiner: Mikael Berndtsson
This project has been submitted by Mikel Urquia Cortabarria for the Bachelor Degree in Informatics to the University of Skövde, in the School of Humanities and Informatics.

Date of submission: May 20, 2013

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

__________________________________________________________________________

Mikel Urquia Cortabarria
I wish to thank, firstly, the supervisor of my thesis in the University of Skövde, Henrik Gustavsson, for providing me valuable technical knowledge and personal assistance.

I wish to thank too my examiner in the thesis, Mikael Berndtsson, whose flexibility and comprehension have been essential to go through with every complication that has arisen during this work.

I am indebted to my family and my many colleagues whose support has been essential to carry out this work, particularly my brother, who has encouraged me to study Computer Science in Sweden.

I would especially like to thank Imanol Mugarza for his personal support during all the academic year. His support has been a great source of energy that has helped me achieving a final result.
Abstract

At the moment, two main web platforms have the monopoly of web service business; .NET and J2EE have been competitors in this area for many years. Within last years, a technological advance has occurred with the appearance of Mono, an open source project that allows .NET technologies to be taken into operating systems other than Microsoft Windows. This opens an information gap that needs to be solved with a new and actualised performance analysis.

This thesis work identifies the performance characteristics of the web platforms on heterogeneous systems. The aim of this study is to investigate different performance characteristics of .NET and J2EE web services in heterogeneous systems. The student systems are Windows 7 and Ubuntu Linux.

A set of web services is built following different service structures, which are then exposed to some quantitative and qualitative test following predefined criteria. The results demonstrate that both .NET and J2EE are suitable web platforms under different circumstances, based mostly on the communication protocol and operating system. This work identifies the best combination of web platform and operating system for each of the web service structures, which can vary for each company.
# Contents

1. Introduction................................................................................................................................. 1  
   1.1 Purpose and situation of this report ..................................................................................... 1  
   1.2 Target readers ....................................................................................................................... 2  
   1.3 Overview of the report .......................................................................................................... 2  
2. Background................................................................................................................................... 3  
   2.1 Usage of the Internet ............................................................................................................ 3  
   2.2 Application integration ......................................................................................................... 4  
   2.3 Software architectures ......................................................................................................... 4  
   2.4 Web Services ....................................................................................................................... 8  
      2.4.1 XML-RPC ....................................................................................................................... 10  
      2.4.2 SOAP .......................................................................................................................... 10  
      2.4.3 WSDL .......................................................................................................................... 12  
      2.4.4 UDDI ........................................................................................................................... 12  
      2.4.5 RESTful Web Services ................................................................................................. 12  
   2.5 Web platforms ..................................................................................................................... 14  
      2.5.1 Java 2 Enterprise Edition (J2EE) .................................................................................. 14  
      2.5.2 Microsoft .NET ............................................................................................................ 15  
      2.5.3 J2EE vs. .NET: The eternal competition ................................................................. 16  
      2.5.4 Mono: Taking .NET into GNU/Linux ....................................................................... 16  
3. Problem ...................................................................................................................................... 18  
   3.1 Problem Statement .............................................................................................................. 18  
   3.2 Objectives ............................................................................................................................ 19  
   3.3 Method ................................................................................................................................ 19
4. Implementation ................................................................................................................................. 21
   4.1 Test Configuration .......................................................................................................................... 21
      4.1.1 Quantitative tests ....................................................................................................................... 21
      4.1.2 Qualitative tests .......................................................................................................................... 22
   4.2 Web Service Applications ............................................................................................................. 23
      4.2.1 Application Architecture Overview ............................................................................................ 24
      4.2.2 Development Tool ....................................................................................................................... 24
      4.2.3 Programming Language ............................................................................................................. 25
      4.2.4 Servers ....................................................................................................................................... 25
      4.2.5 Web Services .............................................................................................................................. 25
      4.2.6 Web Service Clients ................................................................................................................... 29
   4.3 Environmental Settings .................................................................................................................. 33
      4.3.1 Hardware Configuration ............................................................................................................. 33
      4.3.2 Operating System Configuration .................................................................................................. 33
      4.3.3 Network Configuration ................................................................................................................. 33
      4.3.4 Software Configuration .............................................................................................................. 34
5. Result analysis ...................................................................................................................................... 35
   5.1 Quantitative results ............................................................................................................................ 35
   5.2 Qualitative results ............................................................................................................................ 39
   5.3 Overall picture .................................................................................................................................. 40
6. Conclusions .......................................................................................................................................... 42
   6.1 Related Work .................................................................................................................................... 43
   6.2 Future Work ..................................................................................................................................... 43
7. References ............................................................................................................................................ 45
8. Appendices ............................................................................................................................................. 48
   8.1 Appendix A: Abbreviation List ......................................................................................................... 48
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application integration before architectures</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>SOA structure</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Service Oriented Architecture scheme (Image based on: Sun Microsystems, 2013)</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Web Service Architecture (Based on Dodani, 2005)</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>A SOAP message</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>A SOAP service example</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Client-Server constraint</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>J2EE Structure</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>.NET Structure</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>Summary of tests</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Architecture overview</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>SOAP service on NetBeans</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>REST service on NetBeans</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>SOAP service on Visual Studio</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>REST service implementation in Visual Studio</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>REST service definition in Visual Studio</td>
<td>29</td>
</tr>
<tr>
<td>17</td>
<td>Test structure tree</td>
<td>29</td>
</tr>
<tr>
<td>18</td>
<td>Test combination graphic</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>VS client - parameter window</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>VS client - result window</td>
<td>31</td>
</tr>
<tr>
<td>21</td>
<td>NetBeans client - parameter window</td>
<td>31</td>
</tr>
<tr>
<td>22</td>
<td>NetBeans client - result window</td>
<td>32</td>
</tr>
</tbody>
</table>
Figure 23: Hardware information................................................................. 33
Figure 24: Software information ................................................................. 34
Figure 25: Test Set 1 parameters................................................................. 35
Figure 26: Throughput ............................................................................... 35
Figure 27: CPU Usage ............................................................................... 36
Figure 28: Response Time ....................................................................... 36
Figure 29: Test Set 2 Parameters................................................................. 37
Figure 30: Response Time ....................................................................... 37
Figure 31: Throughput ............................................................................... 37
Figure 32: CPU Usage ............................................................................... 38
Figure 33: Response Time Evolution - REST based services ................. 38
Figure 34: Response Time Evolution - SOAP based services ............... 38
Figure 35: Performance table for each test............................................... 41
1. Introduction

Nowadays, almost all the developed world depends on the phenomena known as the Internet and the web. Many Internet and Web technologies such as web services have risen to revolutionize the world as we knew before. It is clearly shown that Web development technologies are a trending topic for the next decades, and therefore many companies are working hard to survive in the web technology race, which does not seem to finish yet. It is a race where a new technology can get old in only a couple of years as new technologies are appearing every time. The users and companies, even if they are getting benefits from all this competitive feeling, they are suffering many troubles due to the extremely fast evolution of the technology.

1.1 Purpose and situation of this report

World Wide Web Consortium or W3C (Haas & Brown, 2005, pp.7), defines web services as “software systems designed to support interoperable machine to machine interaction over a network”. These software systems are multiplatform, and do not depend on the platform or language on which they have been implemented. Thus, several platforms can be used in order to create web services, each of them with its advantages and disadvantages.

At the moment, web service development monopoly is disputed by two big companies: Sun Microsystems and Microsoft Corporation (Miller, 2003, pp.65). Both companies provide technological solutions to develop web services. On the one hand, Sun Microsystems provides a set of specifications known as J2EE (Java 2 Enterprise Edition), which is being implemented by several companies. On the other hand, Microsoft Corporation provides a set of technical solutions known as .NET. Even if both have some common characteristics, they have also very significant differences.

One of the main differences, among several others, is that Sun Microsystems’s solution is fully portable, meaning that it can be implemented and taken into several operating systems, while the solution provided by Microsoft is tied to its operating system, Microsoft Windows. This has been a key fact that business managers and developers have had to take into account when choosing between the two, depending on the heterogeneity of the businesses’ systems. A company may prefer to have a unique and unified operating system on every machine, but that is not the actual reality. Even if all the machines within a company may run the same operating system, an employee that wants to develop a web service application using its personal computer might not be able to do it if it is running a different operating system. This boundary has created a significant distance between .NET and J2EE when it comes to portability capabilities since companies running heterogeneous systems prone to choose a compatible solution for their needs and therefore make use of J2EE.

However, the situation has changed within last years, and the mentioned boundary has been getting thinner due to technological improvements. Mono, an open source project sponsored by Xamarin, allows .NET technologies to be taken into GNU/Linux (McClure et Al., 2012, pp.17-18), avoiding the limitation established by Microsoft.
Therefore, Mono has incremented the possibilities to develop web services. However, despite the high technological offer, there is not enough performance – related information about these. It is impossible to know how each platform performs in different environments, which creates an important lack of information in the area. This report makes a performance comparison of both J2EE and .NET based web services in heterogeneous systems, getting conclusions from the obtained results. These, with the aim of generating valuable results, are based on some criteria taken from similar former studies.

1.2 Target readers

Since this report is intended to provide some useful information about the available technologies and their performance within web services in heterogeneous systems, the target readers of this report are developers and students that are looking for valuable data about the performance of J2EE and .NET based services within different systems. In addition, developers looking for information about application migration will also find this work very interesting, since it will help them to make decisions about their issue.

1.3 Overview of the report

In this report, the reader will get knowledge about web service technologies and the performance of two different web platforms. To achieve this goal, the report has been divided and organized in several chapters that structure the report in an understandable and clean way. To make the division, the author has followed some criteria that can be found at Berndtsson et al.’s work (2008, pp.126-127).

- **Chapter 1: Background material:** This chapter provides background information about the subject area for those readers that are not familiar with it. It explains the evolution of the Internet usages, the creation of business architectures and the actual web service technologies, as well as the actual main web platforms; .NET and J2EE. Already skilled readers should be able to avoid this chapter and start with the further reading.

- **Chapter 2: Problem Description, Aim and Objectives:** Once the introduction to the subject area is properly done, this chapter presents the problem on which the thesis focuses, providing all the necessary information to understand the report orientation properly. An aim resuming the high view goal is also provided. Finally, all the objectives that are meant to be fulfilled are mentioned and explained, followed by the chosen methods.

- **Chapter 3: Implementation of the Solution:** This chapter provides information about the implementation of the solution that has been carried out in order to give a solution to the proposed problem. Contains information about the test configuration and the implemented applications.

- **Chapter 4: Result analysis:** This chapter provides information about all the obtained results, identifying different performance characteristics of the web platforms and finding patterns and relationships between results.

- **Chapter 5: Conclusions:** This chapter contains information about the highlights of the work, future work key indicators and the relationship of the work with former studies.
2. Background

This chapter provides some basic knowledge about the area and sub-area of this investigation. Its aim is not to give very complex and concrete information of any kind, but to help the reader to get a solid basis on which to start reading and comprehending all the content within the thesis. Thus, readers that are already familiar to the research topic should have enough knowledge to skip it and start with further reading.

2.1 Usage of the Internet

The internet has grown exponentially over the last decades from its humble beginnings as a tool created by the USA military in the Cold War, to its present incarnation as a seemingly omnipresent entity that, in the Western world at least, has revolutionized a lot of business and consumer behaviour.

(Harter, 1999)

The internet was, at its beginnings, used to share information between computers and users located in different places all over the network. However, facts like connection speed increase, standardizations... have made it evolve in an incredible speed until today. According to an US SBA (Small Business Administration) article, “Setting up your business on the Internet can be a lucrative way to attract customers, expand your market and increase sales”. This statement has been widely proved by several businesses, and nowadays it is hard to find a company that does not have anything to do with the Internet. It can be a hard task naming more than two big companies that are not making use of the Internet’s advantages. Furthermore, small and medium companies also are moving towards the Internet, deploying web pages and all type of services.

One of the main trends within the last years has been application-to-application communications: these mean automatic connection between two applications inside a business or between different businesses. There can be many examples that can be used to prove this fact; this work makes use of a screwdriver producing company as the ideal one.

A screwdriver producer business might need wood and iron as raw materials to make its own products, but it does not have any resource to extract them, and building them may be too expensive. Instead, the best and easiest solution is to contact a company that actually sells the needed resources (wood and iron). Up to now, the solution seems easy, as the only task of the companies is to contact themselves and make a profitable deal for both of them.

However, from the practical view, the solution is not as easy as it seems at first look. Nowadays, it is not possible to keep a company competitive if all the company-external operations are made manually (calls, paper mails, faxes...). Instead, companies have prone to automatize all the process. To achieve that level of automation, companies started making use of applications to keep track of all their operations, logs, important deals... It was, probably, the first step on the long web services road, which will culminate on a universal application-to-application communications.
2.2 Application integration

At its beginning as a user-faced tool, the web did not support software-oriented interactions very well, and Internet-based applications needed to be able to find, access, and automatically interact with other Internet-based applications (Newcomer, 2002, pp.7).

Applications started to interact between themselves on an individual way, and a kind of unorganized net was being created as lots of application had duplicated functionalities. This was worth enough to answer the needs of the moment, but the way of managing businesses has changed in the last decade; during the post Internet-boom years, cost efficiency has quickly became a key requirement between others, if not the most important one. That means that businesses are being forced to reuse existing systems and applications and to develop new functionalities, constantly adapting to changing business requirements (Krafzig, Bangke & Slama, 2004, pp.15-20).

This new reality was not affordable for the applications and functionalities of the moment. Every needed change, such as adding functionalities, would result on an excessive amount of work, as every duplicated function of every application affected by the update would need to be modified.

As can be seen in the figure 1, each business function had its own network with each section of the data repository. For example, the Service Scheduling function can be connected with CRM, Data Warehouse and External Partner. These sections of the data repository can also be connected to other functions of the business, for example Order Processing and Account Management.

Following the example given before, if a function needed to be changed in the Data Warehouse, as the chosen function would be duplicated three times, the amount of work would increase significantly. A single change might not signify a problem for a company, but if that single change turned into a hundred changes, the situation would be different.

2.3 Software architectures

With the aim of giving an answer to the mess formed by the big amount of duplicated functionalities within companies’ applications, as an evolution of the past attempts to solve the problem, new software architecture concepts were born.
What is *software architecture* as a concept? An erroneous interpretation may turn aside the reader. According to a Krafzig, Banke & Slama’s investigation (2004, pp.76), *software architecture* is a “set of statements that describe software components and assigns the functionality of the system to these components”. *Software architecture* describes the technical structure, constraints and characteristics of the components and the interfaces between them.

Trying to give a solution to the *thousand application mess* described before, several architectures were born, being the following ones the most important ones: *Service Oriented Architecture* (SOA), *Web Oriented Architecture* (WOA), and *Resource Oriented Architecture* (ROA).

**Service Oriented Architecture**

Service Oriented Architecture is a rather hard concept to be defined unilaterally. Everyone has its own definition of the concept made by following personal experiences (Hinchcliffe, 2005, pp.1).

According to Mackenzie et al. (2006, pp.8), “SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains and services as the mechanism by which needs and capabilities are brought together”. Krafzig, Banke and Slama (2004, pp.57) describe SOA as a “software architecture that is based on the key concepts of an application frontend, service, service repository and service bus”. A work made by Rosen et al. (2012, pp.33) define SOA as an architectural style for building enterprise solutions based on services, and it is concerned with the independent construction of business-aligned services. Another definition provided by Hurwitz et al. (2009, pp.5) defines SOA as a “business approach to building IT systems that allows businesses to: leverage existing assets, create new ones and easily enable the inevitable changes required to support the business”.

As mentioned before, defining SOA is not a very easy task. Thomas Erl, one of the SOA luminaries and thinkers and the founder of the web page www.ServiceOrientation.org, highlights eight major technical constraints of SOA that are listed on the following lines (2013, pp.1):

- Service Reusability
- Service Contract
- Service Loose Coupling
- Service Abstraction
- Service
- Composability
- Service Autonomy
- Service Stateless
- Service
- Discoverability

However, Erl does not consider them as SOA constraints but as *Service Orientation* constraints, that are then applied to build *Service Oriented* architectures.

Looking at all the previous definitions, a word appears in the majority of them: service. Services are the main pillar of *Service Oriented Architecture*. SOA has not a unanimous and official accordance between professionals, and many authors have structured it in different manners. According to Krafzig, Banke & Slama’s work’s (2004, pp.75) ideals, SOA could be considered as a design based on four key features or characteristics: application frontend, service, service repository and service bus (See figure 2).
A J2EE and .NET based web service performance comparison

Figure 3 shows clearly how the new architecture works. In the new scenario it is not necessary to make individual connections but all-by accessible services, so that all applications and business functions can access to them in the same manner.

Figure 2: SOA structure

SOA solves the problem shown at the previous section. If a change is needed, instead of changing every function like before, it is enough to update the service that needs the actualisation, a process known as service reuse.

Web Oriented Architecture

Many attempts have been made to successfully implement a SOA within companies, but the level of accessibility and integration has never reached the expected perfection level (Hinchcliffe, 2008). However, there is already a place where the desired integration of systems and information on a large scale has been mostly achieved: the World Wide Web.

Web Oriented Architecture (WOA) is an architectural type based on the immense tensile strength of the World Wide Web and its architectural characteristics. Despite the fact that it has different guidelines than SOA, they remain as close architectures (Hinchcliffe, 2008, pp.1).
WOA is a new concept that has not been inserted in developers mind yet; developers, instead, continue using a SOA. Hinchcliffe (2008, pp.1), one of the thinkers of this new architecture, sets some basic tents that describe WOA characteristics:

- Information in a WOA is represented in the form of resources and are accessed and manipulated via the protocol standard defined in the URI, typically HTTP.
- Every resource on the network can be located via a globally unique address known as URI (Universal Resource Indicator).
- Resources are manipulated by HTTP verbs (CRUD -> Get, Put, Post, Delete) using a technique known as REST (Representational State Transfer).
- Manipulation of network resources is performed solely by the components on the network.
- Access to resources must be layered and not require more than local knowledge of the network.
- The service contract of WOA resources is implicit.
- WOA resources contain embedded URI-s that build a larger network of granular representative state.

As mentioned before, Web Oriented Architecture (WOA) is closely related to Service Oriented Architecture (SOA), as it is a sub style of it that leverages web architecture. It emphasizes the generality of interfaces using five fundamental generic interface constraints: resource identification, manipulation of resources through representations, self-descriptive messages, hypermedia as the engine of application state and application neutrality (Gartner, 2012, pp.1).

**Resource Oriented Architecture**

A resource-oriented architecture or ROA is an architecture that supports internetworking of resources. In other words, it enables the communication of resources within networks (Rouse, 2012, pp.1). ROA is considered a RESTful architecture as it follows the constraints proposed by REST (Representational State Transfer), which are shown in the section 2.4.5 of this thesis report.

Fielding & Taylor (2002, pp.3) identify four different and essential concepts that establish the base of the resource-oriented architecture.

- Resource
- Resource’s names (URIs)
- Representation of resources
- Links between resources.

Moreover, they also identify four properties that a resource-oriented architecture follows:

- Addressability
- Statelessness
- Connectedness
- An uniform interface

ROA is just an implementation of REST principles. Many debates have raise as there is a theoretical division between WOA and ROA defenders. Many of the ROA’s properties are shared within WOA’s
properties, such as the resource utilization or REST principles utilization. Both resources exploit the advantages and facilities the Web Offers (Richardson & Ruby, 2007, pp.80).

2.4 Web Services

All of the three architectures shown before are just architectural styles, and do not represent a concrete method of implementing them. However, a method known as web service has gain renown within last decade, and nowadays is the developer’s preferred one (Bhuvaneswari & Sujatha, 2001, pp.21).

Web services have been defined in several ways by many companies, associations and professionals. One of the main association, World Wide Web Consortium or W3C (Haas & Brown, 2005, pp.7), defines web services as “software systems designed to support interoperable machine to machine interaction over a network”. There are many other definitions for web services, but that is, maybe, the one that encompasses the essential features of them. However, some common conclusions can be extracted from those definitions:

- Services are accessible and can be used utilizing standard Web protocols. Actually, the most used protocol is SOAP (Dodani, 2005, p.22), that it is an XML-based protocol for exchanging information between computers using data transported via HTTP, the most used standard within the web (Cerami, 2002, pp.43). SOAP messages are completely written in XML and transported via HTTP, making it platform and language independent (Cerami, 2002, pp.43). However, a new competitor has risen within last decade, known as REST architectural style, an implementation style (not a standard or protocol) that presents some guidelines to build web services (Ruby & Richardson, 2007, pp.254).

- Services must be self-describing (Cerami, 2002, pp.43-44), so that each new service should have a public service interface that contains information about the service, making it easier to integrate them on client applications. This is usually made by WSDL or Web Service Description Language.

- Services must be discoverable (Cerami, 2002, pp.43-44). Services should be discoverable by other interested parties, being able to locate them and their public service interface (Cerami, 2002, pp.43-44). That is usually made by UDDI or Universal Description Discovery and Integration.

As it can be seen, many protocol and standards appear between the characteristics of web services. Web Service Protocol Stack is a protocol stack that contains most of those protocols. It is used to define, locate implement and make interactions within web services (Bouguettaya & Yu, 2009, pp.121). It can be split into five key layers:
A J2EE and .NET based web service performance comparison

- Communication layer
- Messaging layer
- Description layer
- Discovery layer
- Process layer

One of the principal features of web services is that they do not depend on the platform or coding languages on which were created and deployed; they are platform independent. In other words, web services are multiplatform and multi-language software systems. This is very useful if we think that it is very hard to find two companies that want to connect their applications having the same platform, standards and protocols (Redusers, 2013, pp.17-18).

Another conclusion that can be made from that definition shown before is that web services are the continuation of previous implemented systems on the businesses. Web services are not the first attempt of making machine to machine communications, and so is not a completely new concept but an evolution of a previously existing system. As mentioned before, distributed computing technologies such as CORBA, RMI or DCOM successfully reached this approach.

Stevens (2002, pp.1) states that using web services bring some benefits to the business:

- Reusability
- Location transparency
- Composition
- Scalability and Availability
- Maintaining Investment in Legacy Applications
- Reduced Vendor Dependence

Figure 4 shows the architecture of a web service. As it can be seen, the web service architecture is divided into 3 main objects. The Service provider creates a service, registering it on the Service Registry. In this case, a WSDL protocol is used for that. Once the Service is published on the Service Registry, the Service Requestor requests it, making a search on the Service Registry and finding the needed Service (Dodani, 2005, pp.22).

![Figure 4: Web Service Architecture (Based on Dodani, 2005)](image-url)
Figure 4 is just one way to implement and manage web services, but other ways and protocols can also be used, such as different communication layer protocol (i.e. SMTP instead of HTTP).

### 2.4.1 XML-RPC

XML-RPC was one of the first attempts, together with RosettaNet, on implementing web services across the network, even if at those years the concept was not widely known (Newcomer, 2002, pp.7-9). XML-RPC is a standard protocol that provides an XML and HTTP based mechanism to make function or method calls across network. It is a very simple way to communicate two machines within the network, as it offers a set of simple but useful tools to connect machines and publish machine-readable information (Cerami, 2002, pp.26-27).

XML-RPC calls work with a small XML vocabulary to make and receive requests and responses, respectively. It was created at 1998 by UserLand Software, and it has remained relatively stable since then (Cerami, 2002, pp.26-27).

There are, according to Cerami (2002, pp.26-27), two main ways of using XML-RPC functionalities. On the one hand, it is used to make a kind of glue of two codes inside a private network, solving some of the integration problems a company may have.

On the other hand, XML-RPC can be used to offer services to the world. It is similar to the traditional way on which web was published to humans. It can, therefore, be used by any client that understands an XML-RPC interface.

### 2.4.2 SOAP

SOAP is an XML based specification and protocol, designed to provide the possibility to transport messages between endpoints. It is true that web services would not exist without services and functions, but SOAP is the one in charge of providing the possibility to connect endpoints (Newcomer, 2002, pp.24-26).

SOAP supports XML messaging, and instead of using HTTP requests to get HTML-s (like RESTful web services), it sends XML messages using a transport protocol (HTTP) through the network. However, as it uses XML, both sides of the communication (service provider and service requester) need to have XML processing capability (Newcomer, 2002, pp.24-26).

The SOAP specification, which has extended from XML-RPC specification (Newcomer, 2002) defines 3 main parts:

- SOAP Envelope specification
- Data encoding rules
- RPC Conventions
SOAP messages can also be divided into three parts, as it can be seen in the following picture (Figure 5): envelope, header and body (Newcomer, 2002, pp.24-26).

![Figure 5: A SOAP message](image)

Following these lines, an example of a SOAP application is shown (See figure 6). The header contains information about the device to which the message has to be sent, and the body, instead, contains a message about a reminder.

```
<env:Envelope xmlns:env=\"http://www.w3.org/2001/12/soap-envelope\">
  <env:Header>
    <n:broadcastService xmlns:n=\"http://www.xmlbus.com/broadcastServices\">
      <n:list>Smartphone, Email, Local application</n:list>
    </n:broadcastService>
  </env:Header>
  <env:Body>
    <n:Function xmlns:m=\"http://www.xmlbus.com/broadcastServices/">
      <m:send>
        <m:message>
          Edward, you have to be at school at 14:00 to pick up your son.
        </m:message>
      </m:send>
    </n:Function>
  </env:Body>
</env:Envelope>
```

![Figure 6: A SOAP service example](image)

During this introduction to SOAP, it has been mentioned more than once “SOAP application”, “SOAP messages”... however, it is important to clarify that SOAP is just a protocol and not an actual software or implementation. It defines a set of rules on which the XML messages sent via HTTP can be correctly understood (Newcomer, 2002, pp.24-26).
2.4.3 WSDL

The need for a service description exists since web services expose a software-oriented view of a business or consumer function on where applications may interact within the network. To be able to achieve that, potential consumers must have a description of the service they needed next to information about how to interact with them.

WSDL or Web Service Description Language is a specification that defines how a service must be described in a common XML grammar (Cerami, 2002). It was created to describe and publish the formats and protocols of a Web Service in a standard way (Newcomer, 2002, pp.22-24). According to Newcomer (2002, pp.22-24), WSDL elements contain three main features: description of the data, description of the operations and protocol or transport binding.

2.4.4 UDDI

After a service has been properly published, it has to give facilities to be found and used. That is made by the Universal Description, Discovery and Integration registry (UDDI), which was established by an industry consortium to create and implement a web service directory. It was announced by Microsoft, IBM, and Ariba in September 2000. In May 2001 Microsoft and IBM launched the first UDDI operator sites that set UDDI registry live (Cerami, 2002, pp.134-136).

It can be viewed from two different points of view: On the one hand, UDDI is a specification for building a distributed directory of businesses and web services. Data is stored within a specific XML format. On the other hand, the UDDI Business registry is a completely functional implementation of the UDDI technical specifications, and it was launched, as mentioned before, at May 2001 by IBM and Microsoft (Cerami, 2002, pp.136).

UDDI Specification has two main parts: Registration and Discovery (Newcomer, 2002, pp.26-28). The registration part means that companies providing services can post information to UDDI so that other companies can discover it (Newcomer, 2002, pp.26-28).

Newcomer (2002, pp.26-28) and Cerami (2002, pp.134-136) both agree that information within UDDI can be divided into three types: white pages, yellow pages, and green pages.

2.4.5 RESTful Web Services

RESTful Web Services are Web services based on the architectural style known as REST (Representational State Transfer), and are the preferred ones to implant and deploy Resource Oriented Architectures (Richardson & Ruby, 2007, pp.13).

REST, instead, is a software architectural style (so not a protocol or standard) for hypermedia distributed systems like the Web (Navarro, 2006-2007, pp.4). The term REST was coined by Roy Fielding, one of the HTTP specification creators, in his doctoral thesis on 2000. Fielding and Taylor (2002, pp.1) describe REST as a coordinated set of architectural constraints that attempts to minimize latency and network communication while maximizing the independence of scalability of component implementations.
Tilkov (2007, pp.1), instead, defines *REST* as a “set of principles that define how Web Standards, such as *HTTP* and *URIs*, are supposed to be used. Many people call their implemented web services *RESTful* web services when actually they are violating some of its principles (Richardson & Ruby, 2007, pp.107).

*REST* relies on a stateless, client-server, cacheable communications protocol, that almost in all cases the used protocol is *HTTP*, even if it does not have to be necessarily. *REST* relies in the idea of using Web's benefits instead of making complex operations such as the one done by *CORBA*, *XML-RPC* or *SOAP* to connect machines (Elkstein, 2012, pp.1). Furthermore, the Web itself, based on *HTTP*, could be considered as a big *REST*-based architecture. Roy Fielding splits *REST* into 6 major constraints, which some of them have already be mentioned (See figure 7).

- Stateless
- Client-server
- Cacheable
- Uniform Interface
- Layered system
- Code-on-demand

![Figure 7: Client-Server constraint](image)

*REST*'s motivation is to capture all the benefits offered by the Web. The web is the unique distributed application that have achieved to be scalable to the Internet’s size. This success is, more than other reason, due to the use of extensible message formats and standard.

Particularly, the central concept of the Web is a unified *URI* space, which permits the creation of a dense net of links that make the Web so much used. *URIs* are used to describe resources (conceptual objects), and are distributed by messages through the Web, being able to be extremely uncoupled (Navarro, 2006-2007, pp.5).

There are three classes of architectural elements within *REST*:

- Data Elements
- Connectors
- Components

Many people misunderstand the concepts Architectural Style and Software Systems. *SOA*, *ROA* and *WOA* are actually architectural styles, on which businesses (business to business architecture included) can be organized. In other words, they do not have strictly fixed laws about how to implement them, and so each company can implement them in their own way.
Instead, software systems like web services are not just implementation styles but actual working solutions. They are a way to deploy and implement the architectural styles. It is possible to say that software systems are a palpable version of an architectural style. However, it is not possible to compare them conceptually as they are a completely different thing, although they are also complementary between them.

2.5 Web platforms

All structures, standards, protocols and styles brought before are just specifications on how do they have to be built. However, at the real world, developers need some sort of tools to be able to build them.

Many software systems have been created to answer the need of developing tools. However, only two of them have highlighted between all, and are known as Sun Microsystems’ J2EE and Microsoft’s .NET (Miller, 2003, pp.65). These are the preferred developing platforms by most developers. However, they have several differences that are interesting to highlight. These development platforms are used within different operating systems, from which it is possible to detect some major ones: Microsoft Windows and GNU/Linux.

2.5.1 Java 2 Enterprise Edition (J2EE)

J2EE is a set of specifications (Hunt & Loftus, 2007, referencing to Shannon, 2001, pp.19) and applications programming interfaces (APIs) that builds on top of the J2SE (Java 2 Standard Edition). J2SE provides APIs that are appropriate for the development of standalone applications. J2EE provides some extra support for server-side enterprise applications (Hunt & Loftus, 2007, pp.19).

J2EE is a Java platform designed for the mainframe-scale computing typical of large enterprises, and it was designed by Sun Microsystems to simplify application development in a thin client tiered environment (Rouse, 2005, pp.1).

Sun defines J2EE as a standard for the development of multi-tier business applications. Unlike .NET, J2EE only supports Java as programming language. Figure 8 shows the architecture of J2EE.

Applications made on J2EE can be divided into two, three or more layers. The first layer contains interfaces such as JSP Pages, Servlets and Applets. The second layer, instead, contains EJB components, web services and all business logic. Finally, the last layer is to access databases (Santos, Portilla & Méndez, 2009, referencing to Garrido, 2006, pp.126).
J2EE applications are typically compiled in an intermediate language known as bytecode, which is later interpreted or compiled to native code by the Java Virtual Machine. The JVM behaves as a bridge that understands both the bytecode and the system on which it pretends to execute the code (Santos, Portilla & Méndez, 2009, pp.126).

2.5.2 Microsoft .NET

Microsoft .NET is a platform created in the year 2000 with the objective of matching to the hard competence to the platform presented by Sun. It is a platform for the development of applications, which integrates multiple technologies that have been appearing through last year, such as ASP.NET and ADO.NET between others. It gives support to operating systems based on windows and the new generations of portable machines (Conesa et al, 2010, pp.14-18).

The .NET environment can be divided into the following parts (Herrarte, 2006, pp.1):

- **.NET Framework**, that it is the work environment of the .NET platform.

- **.NET Languages**: .NET supports several languages, from which stands out C#, a language that was specifically created for .NET technologies. In order to make a language compatible with .NET, Microsoft provides a CLS (Common Language Specification), which defines a set of specifications based on ECMA to make a language compatible with .NET.

- **ADO.NET**, that it is a new database interface. It is important to mention that ADO.NET is not an evolution of ADO but a new completely new interface.

- **ASP.NET** is the technology to create dynamic web pages completely integrated with .NET environment.

Kindly similar to J2EE, the process of execution of applications is also divided into steps. .NET applications are compiled into MSIL or Microsoft Intermediate Language, which is the equivalent of J2EE’s bytecode. After that, still similarly to J2EE process, the CLR or Common Language Runtime interprets or compiles the application to native code, which is then ran by the operating system. This can be seen in the figure 9, the CLR acts between the operating system and the user-version application.
2.5.3 J2EE vs. .NET: The eternal competition

Several studies have been made analysing the strength and weaknesses of each platform. Despite their features and functionalities, there is a key fact that is very important to know in order to understand the motivations of this research work: portability. At the first look, it may look that both platforms follow the same process from the code writing to its execution. However, is in there where the major difference can be found. In order to execute a code in any operating system, this needs to support the JVM or CLR that makes the code readable for the machine. The main difference relies here: JVM can be installed and used in several operating systems, while CLR can only be installed and used in Windows operating systems.

Several discussions can occur when talking about platform independence. In the case of .NET, for example, .NET technologies can be considered independent until CLR. MSIL provides an intermediate language that can be read and compiled into native language by an adequate interpreter for each concrete operating system.

Apart from portability issues, performance of each platform has been calculated multiple times. A studio was made first in 2003 by Doculabs, an independent consulting company. After that, Sun Microsystems made in 2004 a new performance analysis, which had been answered in the same year by Microsoft Corporation. These investigations were, shortly, carried out simulating some concrete web services exposed to high user and data load. Each investigation generated different results, which brought some confusion to the developer world.

2.5.4 Mono: Taking .NET into GNU/Linux

A new free tool package called Mono appeared at the year 2004 (year of the release of the first version known as 1.0). The announcement of Mono was made at 2001 on the O’Reilly’s conference) trying to solve .NET’s CLR limitation. The Mono project offers an alternative for Microsoft’s CLR.

According to Xamarin (2013, pp.1), “Mono is a software platform designed to allow developers to easily create cross platform applications. It is an open source implementation of Microsoft’s .NET Framework based on the ECMA standards for C# and the Common Language Runtime”.

Mono is compatible with several operating systems, from Windows and Linux to consoles such as Wii or PlayStation. According to Xamarin (2013, pp.1), it can be divided in four sections:

- **C# Compiler**: Mono’s C# compiler is feature complete for C# 1.0, 2.0, 3.0, and 4.0 (ECMA).

- **Mono Runtime**: This runtime implements ECMA’s CLI or Common Language Infrastructure. The runtime also provides a JIT (Just in Time) and an AOT (Ahead of Time) compiler.

- **Base Class Library**: The Mono Platform provides a set of classes that provide a solid foundation to build applications. These are compatible with Microsoft’s .NET Framework classes.

- **Mono Class Library**: Mono provides more classes than the ones Microsoft offers, which are useful to create Linux applications.
Mono allows developers and programmers to take the .NET technology into GNU/Linux, among other operating systems. However, even if it is almost fully compatible, mono does not support Microsoft .NET’s WPF, WWF and part of WCF (Xamarin, 2013, pp.1).
3. Problem

3.1 Problem Statement

When talking about J2EE and .NET, one of the main differences among several others is that Sun Microsystem’s solution is portable, while the solution provided by Microsoft is tied to its operating system, Windows; J2EE can be implemented and taken into any operating systems that supports its JVM, while the solution provided by Microsoft is tied to its CLR (Common Language Runtime), that can only be ran in a Windows OS (Easton & King, 2004).

The appearance of Mono has balanced the distance that existed before between the two technologies regarding to portability. However, even if Mono provides a solution for .NET technologies on GNU/Linux, it is not an official product of Microsoft and therefore compatibility and performance problems appear from time to time, due to actualizations on Microsoft products and lack of support for some concrete packages and classes, among other problems (Easton & King, 2004).

This whole new scene within web services in heterogeneous systems makes it an area worth investigating. Mono has opened a new gate for developers that wanted to implement .NET based web services not only on Windows but on GNU/Linux too. Likewise, developers that were used to implement only J2EE based web services have now the opportunity to make use of .NET technology. Thus, developers have now a wide range of possibilities to create web services using either J2EE or .NET (Petersen, 2010, pp.393).

However, at the present there is not enough trustable information about the performance of each platform in different operating systems. Several analyses have been made such as the ones by Sun Microsystem, Microsoft Corporation and Doculabs, which focus on a comparison of J2EE and .NET based services within a single operating system. In most of the cases, the former studies have been performed by platform vendors, which make them, at best, hardly trustable studies.

A common characteristic of these is the lack of information about the performance of each platform in different operating systems. Thus, developers cannot know if it is better – from a performance standpoint – to change the operating system or to change the web platform in a migration or deployment case, which is very important information for anyone that needs to choose an operating system or a platform for hosting an application. A study made by TIOBE Software (2013, pp.1-2) shows that the migration between languages is happening continuously, which implies, in many cases, the migration of web platforms. In the case of Java and C#, for example, the study demonstrates that within the last decade the usage of each language has changed around 10% of the total traffic.

*The aim of this thesis work is to investigate different performance characteristics of .NET and J2EE web services in heterogeneous systems. In this specific case, the performance values are based on response time, throughput, CPU usage and portability, while the chosen systems have been Windows 7 and Ubuntu Linux.*
3.2 Objectives

- Build a set of prototype web services using suitable service platforms: In order to have a material on which to carry out the investigation, a set of applications implementing web services have to be developed.

- Prepare an environment to make all the measurements: The environment of the investigation is a very important fact in order to understand and comprehend all the obtained results. Thus, a dedicated environment has to be set according to the targeted test.

- Define a pattern to configure all the measurements: The obtained performance results depend on a big part on the way these are understood. Thus, it is very important to design a measurement scenario that makes the investigator able to classify the results.

- Make a measurement of the built prototypes in both Windows and GNU/Linux: Based on the measurement pattern and the environmental settings, results have to be obtained from the developed web service applications.

- Make an analysis of the obtained results: Once the results are extracted and classified, the investigator has to make an analysis of the obtained results with the aim of identifying patterns and relationships within them.

3.3 Method

This chapter provides information about the way each objective will be carried out based on previous works made by other authors. This way, the author of the investigation will have a guideline to follow through the investigation. These are the objectives and methods:

**Build a set of prototype web services using suitable service platform**

Based on literature papers on web service platforms (Doculabs, 2003, Sanjeev & Nakka, 2005), a set of web service APIs will be selected based on their suitability for platform independent applications, as defined in those papers.

Furthermore, in order to implement the chosen web service APIs, literature papers provided by Flanders (2009), Kalani & Kalani (2004) and Dai, Mandel & Ryman (2007) will be followed.

An alternative to these methods would be to make use of an existing real application, making a case study. However, this would only be useful to analyse a small concrete situation, which it would not provide very useful information for the rest of the developers.

**Prepare an environment to make all the measurements**

Based on previous researches carried out by Doculabs (2003), Sun Microsystems (2004), Microsoft Corporation (2004), Sanjeev & Nakka (2005), and Oracle (2002) on web service performance, a set of operating systems will be implemented under a common underlying
scenario. This guarantees that each web service will have equal possibilities when making the measurement.

**Define a pattern to configure all the measurements**

In order to define criteria that give validity to the investigation, a set of key measurement facts will be chosen, which will be based on previous researches on web service performance benchmarks such as the ones made by Doculabs (2003), Sun Microsystems (2004), Microsoft Corporation (2004), Sanjeev & Nakka (2005), and Oracle (2002).

**Make a measurement of the built prototypes in both Windows and GNU/Linux**

A measurement of the created web services will be made utilizing an experimental method based on literature papers provided by (Doculabs, 2003, Microsoft Corporation, 2004, Oracle, 2002). A set of web services will be moved between operating systems, putting through benchmarking tests and storing the results according to the criteria set before.

An alternative to this method would be to make a literature survey, investigating existing researches about web service performance and portability. However, this would only be useful to summarize the measurements already made, and it would not provide an updated situation of the present technologies.

**Analyse the results obtained from the experimental measurement**

Based on papers about web service performance measurement (Doculabs, 2003, Microsoft Corporation, 2004, Oracle, 2002), an analysis of the obtained results will be made, making use of graphical tools to get clearer conclusions. This objective will be fulfilled if the obtained conclusions are enough to give an answer to the proposed problem.
4. Implementation

This chapter covers all the information related to the implementation and technical solution of the proposed problem. These include the developed technical solution, the test configuration, and the test environment.

4.1 Test Configuration

This section covers information related to all the necessary steps to configure the tests that will be followed in order to get performance results. These have been ideated based on previous investigations on the matter, such as the ones carried out by Sun Microsystems (2004), Doculabs (2003), Microsoft Corporation (2004), and Radview Software (2013).

Two types of performance tests have been carried out based on Doculabs’s (2003) and Radview Software’s (2013) works:

- Quantitative tests
- Qualitative tests

The graphic below (Figure 10) provides a summary of the test types that have been performed as a result of the work made in this thesis.

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Test Description</th>
</tr>
</thead>
</table>
| Quantitative Test | • Defined to measure the throughput and the average response time of each test of every client as well as the average time of all clients under concrete circumstances.  
                  | • The first set of tests has been setup to simulate a low stress in the server, making requests with a single concurrent user.  
                  | • The second set of tests has been setup to simulate a higher stress in the server, increasing the load against it.  
                  | • User quantity has increased proportionally to the performed stress load.  
                  | • CPU utilization has been measured to analyse the correlation with the applied stress. |
| Qualitative Test | • Designed to test the portability of each platform between several operating systems.  
                  | • The test has been performed porting applications between two operating systems, identifying the errors and applying the necessary changes to make them work. |

4.1.1 Quantitative tests

Quantitative tests determine how well services perform based on some concrete numerical measurements and under some concrete circumstances. These are determined by the request, test and user load. According to Doculabs (2003), quantitative test metrics could be captured as following ones:
• Throughput (requests made per second)
• Response time (seconds per request)
• CPU utilisation (CPU consumption per test)

All the tests have been performed under some predefined parameter values – which are described in the test result chapter – in order to follow a continuous pattern. Measurements have been made increasing the number of concurrent users proportionally.

The developed client applications provide information to get enough amounts of data related to throughput and response time. However, an external tool has been used to calculate the CPU utilisation. In the case of Windows, a performance monitor accessible via command (cmd) has been used. In Linux operating system, instead, the system monitor provided by the OS has been utilised.

**Test details**

A set of tests has been designed, based on the work carried out by Doculabs (2003). Each set of tests tries to identify the platform performance under different workload.

• In the first set, the server is exposed to several requests made by a single user, which is used to measure the performance of the platforms under low stress.
• In the second test set, instead, the stress applied to the server has increased proportionally, identifying when each platform starts degrading its performance. This work considers that the network status may vary the results obtained in the measurements. To avoid an obvious alteration, several measurements are performed.

This work considers related the CPU utilisation and the throughput and response times. A correlation between metrics has been performed.

### 4.1.2 Qualitative tests

Qualitative tests determine the level of portability of each platform when taking them into several operating systems. These have been based on the measurement of needed changes to make the package work in the new environment.

As mentioned in the previous chapters, the utilised operating systems are products of Microsoft Corporation and Linux. The performed tests focus on the software compatibility between operating systems.

In the case of the J2EE platform, the utilised software (NetBeans) is available in both Windows and Linux. Instead, .NET’s development tool (Visual Studio) is only available for Windows, and therefore external software (Monodevelop) has been used. Portability tests analyse whether this software transference affects or not to the performance of the platforms.

**Test details**

Portability support analysis consists on porting all the developed applications between two operating systems (Windows and GNU/Linux), identifying all the possible errors and applying all the necessary changes to make them work properly.
This work contemplates the relativity of the results that can be caused by the versions of the chosen operating systems. Therefore, this work has tried to choose similar development level versions for both operating systems, as explained in the environmental settings chapter.

### 4.2 Web Service Applications

For the purpose of this thesis work, two pair of small applications have been made, which are based on an underlying common scenario. Two web applications expose a set of functionalities in the way of web services, which are then consumed by two client applications. Since the aim of this thesis is to analyse the performance of web services, the applications have been isolated being exposed only to service related operations. Based on a performance investigation carried out by Doculabs (2003), the focus of the developed applications is to analyse how well web services perform under concrete circumstances:

- A high number of concurrent users requesting services.
- A heavy load simulated by multiple requests.
- Portability operations between different operating systems.

As mentioned, the applications have the simplest configuration possible to isolate web service and get their real performance. Thus, no business logic has been included in the tests. This contrasts with the investigation mentioned before, which includes a simple database with several rows of data. A work carried out by Sun Microsystems (2004), however, considers that the existence of an external component such as databases may affect and alter the performance results negatively. Thus, the implementation carried out in this thesis work has been done with the goal of simulating existing database functionalities. This has been achieved including the load work in the service request and not making use of external components.

It is important to clarify the key design facts the applications follow, which have been based, again, on Doculabs’s (2003) investigation.

- Every web service has to be independent from each other.
- Web services have to be written by an IDE compatible with the platforms being investigated.
- The minimum effort of error catching has to be done to maintain the code as simple as possible.
- Web services have to be generated using functionalities such as guides and wizards provided by the chosen IDE, manipulating the code and optimizing the poor written code generated by them.
- Error caption procedures have to be as simple as possible. This means that only critical errors have to be caught and treated, such as connection errors.
- Only basic code has to be generated, avoiding extra effort that could alter the performance results.

This work reviews all the implemented code making sure that all these keys are met. In a similar way, all applications have been reviewed ensuring their equivalence in the configuration, avoiding obvious architecture differences among them.
4.2.1 Application Architecture Overview
As mentioned, two pairs of applications have been made using both J2EE and .NET platforms, respectively (considering Mono, for the purpose of this work, an extension of .NET). Other attempts such as the ones made by Sun Microsystems (2004), Microsoft Corporation (2004) and Doculabs (2003) have been taken into account. These investigations contrast between themselves in many areas. However, several common facts make them useful to be used as guidance material.

Each pair of applications consists in two applications that represent a server – client relationship. The following figure (Figure 11) represents the basic structure the applications of this work follow.

![Figure 11: Architecture overview](image)

As it can be appreciated, no interoperability features have been included, which could bring a performance loss that could alter the performance results. Each service requester or client application makes requests to the service provider or server application, which sends a response after having made all the requested operations.

4.2.2 Development Tool
The chosen development APIs for the creation of web services have been NetBeans IDE for J2EE and Visual Studio 2012 Ultimate for .NET, respectively. MonoDevelop has also been used when taking .NET web services to a GNU/Linux distribution. In the case of the applications based on the J2EE platform, there are several free open source choices that might be useful for the purpose of these applications. However, NetBeans IDE makes itself the best selection since it is available for both Windows and GNU/Linux and offers full support for JAX-WS API. Furthermore, the familiarity of the author of this thesis with the development tool has also affected in the API selection.

In the case of the .NET platform, instead, the solution offered by Microsoft in order to develop web services is Visual Studio 2012. There are also other alternatives that could be used such as SharpDevelop, but since Visual Studio is the most widely used software within companies, it is the best option to simulate most accurately possible the actual business situation.
4.2.3 Programming Language
In the case of J2EE, the unique possibility to develop applications is by making use of Java, the supported programming language. In .NET’s case, instead, there are several possibilities that can be used. In order to get the best profit from .NET, C# (C Sharp) has been used, since it is a programming language specifically designed to make the best use of the .NET platform.

4.2.4 Servers
Since the clients will have to connect with the service providers, installing and running a server is essential. For the purpose of this work, Glassfish for J2EE and IIS Express for .NET have been used with the addition of XSP4 for Mono based applications.

Regarding to Glassfish, since it is an application server developed by Sun Microsystems, it has been chosen against its direct competitor, Tomcat. Although the performance of both is similar (Raible, 2007), Glassfish provides an almost complete support for Java 2EE, while Tomcat is a servlet container only, which does not provide Java 2EE support.

In the case of .NET, instead, there are two options that might be useful for the purpose of this work: IIS Express and the default IIS. IIS Express provides less features than the default IIS, but since it is a commercial product – and therefore not free –, IIS Express has been the most adequate choice. Nonetheless, IIS Express provides enough features to carry out this work, and thus there is not any critical performance loss to be afraid of.

XSP4 server is not exposed to any incompatibilities against IIS Express, which makes a good choice against its competitor, Apache Tomcat. It is important to notice its limitation regarding to HTTP support (it supports HTTP version 1 only), but since the performance measurements have been made in a localhost level (See 4.2 Test Configuration), the issue does not affect negatively to the obtained results.

4.2.5 Web Services
Taking note of other attempts to measure web service performance, such as the ones made by Sun Microsystems (2004), Microsoft Corporation (2004) and Doculabs (2003), several web services have been implemented. All services have been configured so that they can be consumed using both REST and SOAP. This feature has been essential to determine which platform performs better with each communication protocol. All the generated code meets the key design facts mentioned at the beginning of the section.

Both NetBeans and Visual Studio offer facilities to implement web services. The method to deploy and consume them varies in each platform. Therefore, a short explanation about these might be important. However, the functionality of the developed services is the same in both platforms. In order to produce trustable results, four web services have been programmed following previous works on the subject, as showed in the following lines:

- **Request an empty function (emptyRequest)**: This function is no more than an empty service that does not return any value. Therefore, it does not have any type of input value. This service is pretended to simulate a high load of requests in the server application.
- **Request a function with a response (responseRequest):** This function has the same functionality as *emptyRequest*, but returns a string value containing “SOAP/REST response”. This service is pretended too to simulate a load of requests in the server application.

- **Request a function with an input (setParameter):** This service receives a String value as an input, and does not return any type of value. It is intended to simulate load on the server, but focusing on the performance related to parameter transportation.

- **Request a function with an input and a response (getResponseParameter):** This function receives a String value as an input, and returns another String value containing ”SOAP/REST response”. This function focuses on the performance on parameter transportation of both types (reception/return).

The coded web services do not perform any type of internal operation, as they are only useful as long as they work only on the areas focused on the work. In this case, services have to focus on communication performance, and therefore no extra operations are necessary.

The following sections provide information about the implementation of the applications. However, they only make an overview of the same; the complete code of the server and client applications can be found at the annexes.

**Web Services on NetBeans**

*NetBeans* offers facilities to develop both SOAP and REST web services. In order to have full support for the creation of web services, a web application has been created.

In the case of SOAP web services, the platform offers a wizard that guides the user through a small form, creating an example service. In order to implement the desired services in the newly created class, it is enough to replace the existing *helloworld* example with the functions that are needed and delete the unnecessary code generated by the wizard (See figure 12).

```java
package SOAP;
import javax.xml.ws.WebMethod;
import javax.xml.ws.WebService;

@WebService(serviceName = "SoapNS")
public class SoapNS {
    @WebMethod(operationName = "emptyRequest")
    public void emptyRequest() {};

    @WebMethod(operationName = "receiveRequest")
    public String receiveRequest() { return "SOAP response"; }

    @WebMethod(operationName = "sendRequest")
    public String sendRequest(String parameters) {
    }

    @WebMethod(operationName = "completeRequest")
    public String completeRequest(String parameters) { return "SOAP response"; }
}
```

*Figure 12: SOAP service on NetBeans*
REST based web services, instead, are slightly different than SOAP based ones. Last ones are consumed making use of a WSDL file (See introduction chapter), which define what services exist and how are they implemented. In the case of REST, instead, the advantages of the web are used. Unlike SOAP, clients can access directly to the web services via HTTP without making use of definition files.

*NetBeans* offers a wizard to create REST services. These can be developed from scratch as much as from already existing databases. This way, its implementation can be customized for the programmer’s needs.

Similar to SOAP services, the wizard creates a new class with an example REST resource. In order to provide adequate REST services, four resources have been created, one for each service type. The following image shows one of those resources, which implements the `emptyRequest()` service. Note that services are complemented with some syntax; in the case of the shown picture, the syntax determines the method to call the service (`@GET`) and the return format (`text/plain`). However, alternatives such as `@POST`, `@UPDATE` or `@DELETE` could be used, following always the REST principles.

One of the most important facts to be taken into account when implementing REST web services is the path. The path is a vital part of the service consumption. Since the path is the one the client will follow to access the service, it is important to be representative, following the REST principles. Therefore, a resource like the one shown it the following figure (figure 13) has a path that describes the nature of the service, `emptyRequest`, in this case.

```java
package REST;

import javax.ws.rs.*;

@Path("emptyRequest")
public class EmptyRequestResource {

    @Context
    private UriInfo context;

    public EmptyRequestResource() {
    }

    @GET
    @Produces("text/plain")
    public String getEmptyRequest() { return ""; }
}
```

*Figure 13: REST service on NetBeans*
Web Services in Visual Studio

Similar to NetBeans, Visual Studio provides assistance to make SOAP and REST web services. A WCF project has been created, which provides support of the creation of services. In the case of SOAP web services, for example, the process is identical to the creation of SOAP services in NetBeans. Visual Studio creates a hello world service, which is then modified according to the needs.

Apart from some lexical differences, the way how the services are made is significantly similar. Likely, the four services have been implemented in a way that the client will be able to connect and consume them. The following picture (Figure 14) shows the final result for the services class that is implemented in the class named SOAPWebService.asmx.

![SOAP service on Visual Studio](image)

The main difference between both platforms comes when implementing RESTful web services. Although the base process is similar, there is a concept that developers have to take into account; the contract.

Apart from implementing the services, it is essential to define them in an extra class. With that purpose, the REST services assistant creates two files: the service file that contains the service implementation and the class file that contains the definition. Both files have been modified so that the final result looks like the following images. The first one (Figure 15) shows the definition file. The service interface defines how the services are consumed (including the path) and which parameters receive. The second one (Figure 16), instead, shows the implementation of these definitions.

![REST service implementation in Visual Studio](image)
Web Service Clients

Since the objective of the thesis is to analyse the performance of the created web services, only web service related content is explained.

Both clients (developed in Visual Studio and NetBeans) consume the developed web services after having set the parameters that customise the performance tests. These parameters define how the tests are performed, changing load effort size. The list below shows all of them.

- **Request quantity**: Determines the request quantity performed by each test.
- **Test quantity**: Determines the test quantity performed by each user.
- **Parameters (send)**: Determines whether parameters have to be sent to the services or not.
- **Parameters (receive)**: Determines whether parameters have to be received from the services or not.
- **User quantity**: Determines the quantity of users (making use of threads) that have to perform tests.
- **Communication method**: Determines the communication method to consume the service, which can be SOAP or REST

The first four parameters’ goal is to simulate load effort, while the user quantity simulates effort related to multiple user interaction.
Following this architecture (See figure 17), different services are requested depending on the introduced parameters. For example, if the user wants to request a service without parameters, *emptyRequest* is requested. The following graphic (Figure 18) defines the combination options for each request:

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Request N.</th>
<th>Test N.</th>
<th>User N.</th>
<th>Send P.</th>
<th>Receive P.</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>emptyRequest</em></td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>No</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td><em>responseRequest</em></td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>No</td>
<td>Yes</td>
<td>Any</td>
</tr>
<tr>
<td><em>setParameter</em></td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Yes</td>
<td>No</td>
<td>Any</td>
</tr>
<tr>
<td><em>getResponseParameter</em></td>
<td>Any</td>
<td>Any</td>
<td>Any</td>
<td>Yes</td>
<td>Yes</td>
<td>Any</td>
</tr>
</tbody>
</table>

![Figure 18: Test combination graphic](image)

After all the necessary tests have been performed, the clients pop up the response time results in many different ways, as it is discussed in the test configuration section. The results are the following ones:

- Response time for each test (not including response time for each request).
- Average response time of all the tests for a) each user and b) all users.
- Throughput for each test (requests per second)

This makes the reader able to detect where each platform performs better.

**Visual Studio Client**

The developed client on *Visual Studio* consists mainly on two graphical forms, which then call the classes to perform web service requests. The graphical interface contains all the necessary files to introduce the desired parameters described right before (See figure 19).

![Figure 19: VS client - parameter window](image)

Regarding to the consumption of the services, *Visual Studio* offers facilities to make it in an easy way. SOAP services are consumed making use of a service reference, which is determined by the WSDL file generated by the service. REST Services, instead, do not need any type of definition file, and it is enough
to know the path for the resources. In the case of emptyRequest, for example, the correct path would be the following one:

\[
http://localhost:16572/RESTWebService.svc/emptyRequest
\]

All these operations are made in the class OperationThread.cs. Once the requests have been performed, the application shows a new graphical interface, showing the results obtained in the tests (See figure 20).

![Image](image1.png)

**Figure 20: VS client - result window**

**NetBeans Client**

The developed client on NetBeans, similar to the one made on Visual Studio, consists on a frame and a dialog, which then call the classes to perform web service requests. The image below (Figure 21) shows the appearance of the frame window.

![Image](image2.png)

**Figure 21: NetBeans client - parameter window**

The consumption of the web services in NetBeans is similar to Visual Studio. In the case of SOAP services, the wizard provided by the software creates a web service client from the wsdl file provided by
the user. In the case of REST, instead, the process is identical to Visual Studio REST clients. The user has to introduce a valid direction to the REST service. Comparing the URL with the one described in the Visual Studio section, both URL-s are very similar.

http://localhost:8080/WebServer/webresources/emptyRequest

All these operations are made in the class OperationThread.java. Once the requests have been performed, the application shows a new graphical interface, showing the results obtained in the tests (See figure 22).

![Figure 22: NetBeans client - result window](image-url)
4.3 Environmental Settings
This chapter provides relevant information about the environmental settings used to perform the measurement tests, based on related works made by Sun Microsystems (2004), Microsoft Corporation (2004), Doculabs (2003) and Sanjeev & Nakka (2005). These settings include:

- Hardware Configuration
- Operating System Configuration
- Software Configuration
- Network Configuration

4.3.1 Hardware Configuration
A single laptop machine has been used to perform the measurement tests indicated in this work. The following list shows a detailed view of the features of the same (Figure 23).

<table>
<thead>
<tr>
<th>System Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System model</strong></td>
</tr>
<tr>
<td><strong>CPU quantity</strong></td>
</tr>
<tr>
<td><strong>Processor</strong></td>
</tr>
<tr>
<td><strong>Installed physical mem. (RAM)</strong></td>
</tr>
<tr>
<td><strong>System type</strong></td>
</tr>
</tbody>
</table>

The whole investigation has been performed in a single machine. Having run both platforms on the same machine, it may have affected to the results values. Usage of several machines would be a more appropriated configuration. However, due to the resource limitation to which this thesis work is exposed, it has not been possible to handle multiple machines.

4.3.2 Operating System Configuration
In order to perform the portability tests indicated in the previous section, two operating systems have been utilized within the configuration: Microsoft Windows 7 Home Premium and Ubuntu 13.04 Linux.

Two partitions have been made on the machine, installing one operating system on each. In order to maintain the equivalence between systems, the – providers – recommended updates have been applied, which have mostly been operating systems’ actualisations. This reflects the situation of the real world businesses. The utilised versions are reflected on a summary table on following sections.

4.3.3 Network Configuration
Due to lack of resources, a complete network infrastructure was not available for the purposes of this work. Therefore, all tests have been performed within a single machine, which acts both as server and as client (with the localhost direction). This may have increased the performance results slightly, which
could not represent accurately the real performance results. However, that depends entirely on the network configuration and it does not rely on the tested platforms.

4.3.4 Software Configuration

Obtained performance results rely partially on the version and type of the software that has been utilised. Therefore, equivalence in software versioning has been maintained, assuring the credibility of the results. The following table summarises all the version-related information (Figure 24).

<table>
<thead>
<tr>
<th>Software Version</th>
<th>Version/Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 7 Home Premium</td>
<td>6.1.7600 Compilation 7600</td>
</tr>
<tr>
<td>Ubuntu Linux</td>
<td>13.04 (last stable version)</td>
</tr>
<tr>
<td>Visual Studio 2012</td>
<td>11.0.60315.01 Update 2</td>
</tr>
<tr>
<td>.NET Framework</td>
<td>4.5.50709</td>
</tr>
<tr>
<td>IL Express</td>
<td>7.5</td>
</tr>
<tr>
<td>NetBeans IDE</td>
<td>7.3 Build 201302132200</td>
</tr>
<tr>
<td>JRE (Java Runtime Environment)</td>
<td>1.7.0_21-b11</td>
</tr>
<tr>
<td>Glassfish server</td>
<td>3.1.2</td>
</tr>
<tr>
<td>Mono</td>
<td>2.10.8.1 (NET 4.0)</td>
</tr>
<tr>
<td>Monodevelop</td>
<td>3.0</td>
</tr>
<tr>
<td>XSP4 Server</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 24: Software information
5. Result analysis

This chapter provides all the obtained results showed both graphical and literally. Results are showed according to the test configuration, showing results for each and all test set. These include:

- Quantitative results
- Qualitative results

5.1 Quantitative results

The following results demonstrate that each web platform have a better or a worse performance depending on the communication protocol and the operating system. Regarding to SOAP services, for example, .NET has a slightly better performance in a low-medium stress situation, while J2EE has the advantage on REST based services.

The choice of the OS has also an impact on the performance values. With a low stress value, the performance on each OS is very similar, but under a higher stress pressure, the performance starts to vary. In fact, services deployed on Linux have a better performance comparing to services deployed on Windows.

Test Set 1

The parameters for this tests set have been (Figure 25):
Findings

Under a low level of stress, several results have been obtained, which can be analysed based on two main categories: communication protocol and operating system.

From the SOAP perspective, it is possible to observe that .NET responds slightly faster than J2EE. Indeed, .NET based services are almost two times faster than J2EE based services. This demonstrates that the integration of SOAP in .NET is higher than J2EE. However, the usage of the CPU is also higher in .NET than J2EE, which could be attributed to the higher performance of the first. It is important to notice that results do not alter significantly between different systems except of the CPU usage; Linux consumes slightly more CPU resources than Windows (around 12% more).

Regarding to REST results, instead, J2EE based services have a much higher throughput than .NET based services. J2EE services responds more than 8 times faster than NET services in both operating systems. A curious paradox can be found in the CPU usage: the CPU usage is bigger for J2EE services in Linux, while the same happens to NET services in Windows. Regarding to OS performance differences, it is possible to observe that the overall performance is higher for Windows comparing it to Linux.
Test Set 2
These have been the parameters for the second set of tests. The user quantity has been increased constantly by ten on each test series until the number of 500 has been reached. The other parameters have not been exposed to a continuous increment of their values (See figure 29).

<table>
<thead>
<tr>
<th>Test Details</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Number</td>
<td>50</td>
</tr>
<tr>
<td>Test Number</td>
<td>10</td>
</tr>
<tr>
<td>User Number</td>
<td>1–500 (increment of 10)</td>
</tr>
<tr>
<td>Send Parameters</td>
<td>NO / “ParametersExample”</td>
</tr>
<tr>
<td>Receive Parameters</td>
<td>Yes / No</td>
</tr>
<tr>
<td>Communication Protocol</td>
<td>SOAP / REST</td>
</tr>
</tbody>
</table>

Figure 29: Test Set 2 Parameters

Results are showed according to the used communication protocol and operating system. The evolution of the performance is also displayed.

Figure 30: Response Time

Figure 31: Throughput
A J2EE and .NET based web service performance comparison

Figure 32: CPU Usage

Figure 33: Response Time Evolution - SOAP based services

Figure 34: Response Time Evolution - REST based services
Findings

A couple of affirmations can be made taking a look to the results. These vary significantly as the quantity of users increase. Similar to the low stress tests, the results can be sorted by the communication protocol and the operating system.

On the one hand, it is possible to say (referring strictly to communication speed) that .NET is still the best choice for SOAP based services. The performance of NET based services is slightly higher in both Linux and Windows. However, the response time evolution chart shows that there is not any available data from a concrete number of users. It can be considered a paradox: NET services are faster than J2EE with low – med stress, but they crash for extreme stress values. It is important to notice that the crashes have happened more often in Linux than Windows operating system, which could be attributed to Mono compatibility issues.

On the other hand, J2EE is shown to have the highest performance for REST services. J2EE based services are two times faster than .NET services, which makes it a better choice. In addition, Mono does not fully support REST service, and crashes have happened constantly from 10 users. However, crashes for NET have also happened in Windows from 400 users.

Regarding to the CPU usage, .NET based services makes a considerably slower use of CPU resources. While NET services have ranged mostly between 60%-80% of the CPU resources, J2EE services have reached the maximum capacity considerably fast, being Linux the most consuming operating system.

On a final note, it is of vital importance the performance difference between operating systems. The obtained results demonstrate that services deployed on Linux are considerably faster than services deployed on Windows. SOAP based services, for example, are two times slower on Windows (regarding to response time), which is a considerable difference taking into account that the analysed items are the web platforms instead of the operating systems.

5.2 Qualitative results

The following results demonstrate that although Mono provides a high level solution for .NET based services in heterogeneous systems, J2EE still has the superiority regarding to portability. After making some important changes, Mono has been able to run .NET based services, but still several errors have happened. Indeed, Mono has not been able to perform successfully all the performed tests, and several incompatibility problems have appeared.

J2EE based services, instead, have been proved as fully portable applications, and no changes have been necessary in order to run the services successfully.

Test Results

These are the changes applied to each web platform application:

.NET based services

The first trial of porting .NET based application has been trying to open-it directly from Monodevelop. However, the web configuration structure in Monodevelop is not the same as the one in Visual Studio.
Therefore, a new web application has been created, importing all the necessary files. The following list shows all the followed steps to make the applications work.

1- Install XSP4 server
2- Create a new web application
3- Import service files (SOAP and REST services)
4- Change .NET framework from 4.5 to 4.0 (4.5 is not fully supported yet)
5- Change web.config file. Insert service description and service behaviour (for REST services)
6- Set fixed port and IP (127.0.0.1:8080)

Since some changes have been made in the server side of the applications, some minor syntax changes have been done in the application. These are related to the re-definition of SOAP client, actualisation of REST directions. However, this work does not try to analyse client’s performance, and therefore the performed changes are not taken into account.

It is important to notice that although the applications were able to be ran, several error occurred due to incompatibility issues, which can be observed in the performance test charts. Concretely, the server did not support more than 10 concurrent users consuming REST services. The support for SOAP services was slightly higher, supporting service consumption until approximately 200 concurrent users.

J2EE based services

The results obtained from the test demonstrates that J2EE based service applications are fully portable. No changes have been needed to make the code work on Ubuntu Linux. Indeed, it has been enough to take the application and running it in the new operating system.

Furthermore, no errors have occurred; all tests have been performed without any unwanted errors.

5.3 Overall picture

Taking a look to the individual results obtained in each type of performance test, it is difficult to define how well each platform performs from a high perspective. In order to find patterns and relationships between the results, a common activity is to construct a table displaying a summary of all the obtained results. The following figure (Figure 35) shows which platform has given better results for each performed test.

As it can be appreciated, some patterns can be found regarding to communication protocol. As discussed in the conclusions chapter, each platform has resulted better with a specific communication protocol. In the case of SOAP based services, for example, applications developed with .NET have given better results than applications implemented with J2EE. In the case of REST based services, instead, the situation is the opposite; in fact, J2EE shows to perform better than .NET.
However, despite this chart provides an overall summary of the results, there are many other parameters such as the operating system that are not included. The following chapter discusses the performance based on all the investigated parameters.
6. Conclusions

In this thesis work, two different web platforms have been investigated, analysing their performance in different environments. The analysed platforms have been the J2EE specifications by Sun Microsystems and the .NET technologies by Microsoft Corporation, while the investigation has been performed in a Windows and a Linux environment. The results obtained in this work demonstrate that there is not an absolute answer for web platform related questions. Each of the analysed platforms has the advantage in different areas.

According to the results obtained from the tests made in this work, .NET stands as the best solution for SOAP based services from a performance standpoint, while J2EE is a better choice for REST based solutions. It is important to notice that the obtained performance difference between platforms has been considerably big in every performed test; this proves that each web platform is optimized for a concrete service method rather than for all of them.

In terms of CPU consumption, J2EE makes a higher use of the available resources, while .NET tries to rationalize them. The portability support of the web platforms has also been investigated in this work. According to the obtained result, it is possible to observe that J2EE still is the best choice to port applications between operating systems. Mono has been proved as an excellent alternative for .NET in Ubuntu Linux, but several incompatibilities still exists.

An interesting result has been that the choice of the operating system has a great impact on the obtained performance values. Windows 7 has stand as the best operating system for REST based services, while Ubuntu Linux has a higher performance for SOAP based services.

This work demonstrates that the response for the question presented in the beginning of this work may vary significantly. It is very hard to give a unique identical answer for every case of migration within web service applications. However, although a definitive answer is not viable, a set of tips and guidelines can be extracted from the obtained results.

The concluded tips have been based on the performance and stability of the web platforms. Thus, three different scenarios have been identified, which depends on the communication protocol.

Regarding to REST based services, the analysis of the obtained results shows that the best solution from a performance and stability standpoint is a combination of J2EE and Windows. As far as SOAP is concerned, the best combination would be to make use of .NET and Linux.

However, it is important to notice that in many cases both communication protocols might be used jointly (e.g. Amazon web services). In that case, there is not a best combination; instead, the developers have to make an investigation of what platform would fit best their needs based on the results obtained in this work.

Taking a look to the contributions this work could do to the society it can be observed that it is an investigation that could perfectly define a pattern on the web platforms market share. This work helps companies taking technological decisions based on real statistics rather than personal and external experiences. On a side effect, web platform owners would stop making strong marketing (which would lose real value) and would try improving the weaknesses identified in this work. This, of course, would
result on an improvement in web technologies industry. All the parameters investigated in this work would be exposed to an increase of the performance (throughput, performance time, CPU usage, portability support).

As a final note, future investigations could use this work and the tools developed in the same in order to contrast and observe how well or bad the evolution of web platforms have been.

### 6.1 Related Work

An interesting activity during a thesis work is to compare the obtained results with former studies on the investigated area. In this specific case, making this comparison is only partly possible; there are some limitations that have to be taken into account.

On the one side, most of the investigations on this area have been carried out by the vendors of the target platforms. Thus, the investigator has to be aware of the relativity of these investigations.

On the other side, most of the performed investigations are based only on SOAP based services, and thus it is hard to identify references for REST based services. Therefore, only references based on SOAP services have been possible to be found.

Having considered these limitations, some relationships have been found between this work and former studies. In fact, the direction of the results based on SOAP services matches with the investigations made by Microsoft Corporation (2004) and Sanjeev and Nakka (2005). In addition, the difference between J2EE and NET performance has also been similar in all the investigations. This could be attributed to the similarity of the implemented services, which do not differ excessively much.

### 6.2 Future Work

This work has been useful in order to get some major facts about the performance of two concrete web platforms in heterogeneous systems. However, it could be enriched adding some other features.

On a personal note, if I could have the opportunity to keep working on this investigation, I would like to develop new features to implement to the service / client applications, which are the following ones:

**Client Application Improvements**

- Export results to selected to a selected *Microsoft Excel / Openoffice Calc*
- Save and load user created test pre – sets
- Add interoperable service requests
- Log error frequency on tests
- Automatize stress increase process

**Server Application Improvements**

- Introduce more service types (internal operations, delays, connection limit...)

43
• Identify and display origin of the occurred errors

In addition, I would like to go deeper on each platform analysing the performance from different points of view such as support for external libraries and support for different new technologies. In addition, the portability state of NET could change with a proper release of a new Mono version, which it would highlight the need for an actualised investigation.

A weak point of this work might be that it has been focused on two platforms instead of more. Due to manpower and resource limit, it has not been possible to extend the investigation to other web technologies. There is a vast amount of web technologies to be investigated that it could be an immense work for a single investigator. However, they deserve a place on this investigation as it could help to encourage competitive feeling between the platform owners. Big companies interested on the potential results of this kind of investigation could have enough reasons to extend this work to new areas.
7. References


8. Appendices

8.1 Appendix A: Abbreviation List

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>Common Language Runtime</td>
</tr>
<tr>
<td>CLS</td>
<td>Common Language Specification</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Beans</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MSIL</td>
<td>Microsoft Intermediate Language</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>ROA</td>
<td>Resource Oriented Architecture</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WCF</td>
<td>Windows Communicatino Foundation</td>
</tr>
<tr>
<td>WOA</td>
<td>Web Oriented Architecture</td>
</tr>
<tr>
<td>WPF</td>
<td>Windows Presentation Foundation</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WWF</td>
<td>Windows Workflow Foundation</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>Extensible Markup Language - Remote Procedure Call</td>
</tr>
</tbody>
</table>