FRAMEWORK FOR DYNAMIC RECONFIGURATION OF WEB SERVICES

Jyotirmay Sarna

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand

Abstract

All software requires maintenance, even onc that runs over networks. We have designed and implemented a way to do scheduled maintenance on Web Services without forcing any downtime. The means is to dynamically switch Web Services at runtime with a backup Web Service, so that the clients face no interruptions and the maintainers can upgrade the original Web Service. In the research phase two key things were established. Firstly, that an external application (manager) used by the "maintenance administrator" would be needed to initiate Web Service switching. Secondly on the Client side, an interceptor layer will be needed to manage the extra functionality introduced by Dynamic Reconfiguration. Based on this understanding, we designed and implemented two frameworks called Interrupt based and Exception based design. Interrupt based is where the Manager interrupts the client with a new Web Service address. In Exception based, the Client's request to the Web Service is responded to with an exception, which triggers its interceptor layer to connect to the UDDI (Online Web Service Address directory) and get the new Web Service address. This entire process happens in under 0.2 seconds for both of our frameworks.

1. Introduction

The aim of this project is to work towards making dynamic reconfiguration of Web Services a reality. This project was undertaken by Gene Lee and me to fulfill the project requirements for the final year of our Engineering degree. The project was commissioned by the University of Auckland's Software Engineering department and supervised by Ian Warren. To begin with the two keywords in our aim, Web Services and Dynamic Reconfiguration are briefly explained below. After this I will present our motivation for undertaking this project, followed by the project objectives we formulated and finally I will close this section with an outline of the remaining report.

Web Services is a recent development, materializing in the last decade. Web Services are functional entities that can be invoked over a network using internet based protocols. They allow software and services from different companies and locations to be combined easily to provide an integrated service [1]. These benefits make the utilisation of Web Services based architecture an attractive option in internet and other network based services.

Dynamic Reconfiguration is the process of making changes to an executing system without requiring that the system be temporarily shut down. It is typically used in systems which cannot afford to be temporarily shutdown and at the same time are expected to deal with change in their working environment [2]. A typical scenario where it is used is when a mobile phone user's call is transferred between adjacent communication towers, as he/she moves from one tower's area of influence to another's. Despite this change, the connection or quality of the mobile user's call is not affected.

1.1. Motivation

Web Services can be imagined as being the components that constitute a network-based or online service. The society today expects online services to be provided continuously, reliably and conveniently especially where online services have carved up a niche for themselves, in areas of commerce, which need these qualities. This expectation has led companies to build their business models around the availability of these online facilities at all times. Examples of such commercial applications include online banking, ATM's, online auctions etc. If these services go down for maintenance, even for a short while, their owners loose a lot of money and customer faith.

Yet online services like any other software require scheduled and often unscheduled maintenance. During this period, the service has to become unavailable for a period of time, causing loss to the company. Online auctions are an example of an online service that cannot afford such downtime. Many businesses have formed, which use the online auctions to do a large proportion of their business on a daily basis much like online banking and search engines. When in June 1999, e-bay online auctions suffered outages, the longest one lasting 22 hours; the company lost 25% of its stock price, costing e-bay almost $US five million. In addition, the company lost face as the incident was publicised on CNN. To make matters worse, ever since even a short outage brings media attention and a consequent fall in e-bay's stock price [3] effecting shareholder and customer confidence.

Since connections to a Web Service exist over networks, it maybe argued that incoming requests to the

server running the Web Service could be redirected to a server running the backup web service by placing a router or a dynamically reconfiguring FPGA device on the connection. Even though this will solve the immediate problem, but it is not a complete solution, especially if the backup service does not exist within the sub-network of the router.

1.2. Project Objectives

Thus we defined our objectives to be firstly, to design frameworks that will allow dynamic reconfiguration of Web Services. Secondly, implement these frameworks in a suitable Object Oriented language. Thirdly, to gather data for their affect on memory usage and response time under test in a real world scenario. Fourthly, to compare the performance of the two designs with each other based on the data collected in the last step. And finally to evaluate properties of their design based on dynamic reconfiguration criterion, which we would have to research.

1.3. Outline

The rest of the report will focus on initially providing the reader with background information required to understand the underlying technologies involved and develop a criterion to judge the solutions. Then the designs for our solutions will be presented with a discussion on why we made those design decisions. Next, the designs will be put to test and compared with each other and evaluated for their memory usage, response time and other design features. The report will end with revisiting of the objectives and the key findings of the report.

2. Research

This section contains all the Background information required to create the understanding for the reader to understand our solutions.

2.1. Web Services

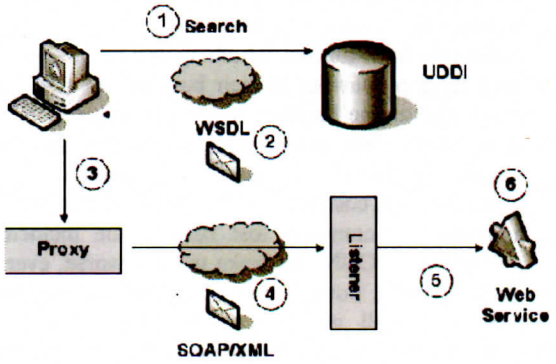


Figure 1: Detailed process flow of a Web Service [4]

The complete interaction between a Web service and its requestor can be divided into six logical steps.

The first three being discovery, description and proxy creation, which define the connection establishment phase. Here the client application searches the UDDI for a Web Service that satisfies its requirements. UDDI (Universal Description, Discovery and Integration) is an online registry of available Web Services. The client examines an XML document called WSDL with information on the Web Service functionality and communication protocols, to determine the suitability of the web service to its needs. Having chosen a Web Service, the client application binds itself with the Web Service through the creation of a proxy. The Proxy acts like the middleman between the client and the Web Service, handling all network concerns in their communication.

The remaining three steps define the normal operation of the client Web Service interaction. This is where the proxy turns the client's method invocation into an XML message and gets it over the network to the Web Service address. Here the Web Service is actively listening for messages. On receiving the message, it decodes the XML message and performs the request locally, returning the result to the sender again in XML format [4].

2.2. Problem with Web Services

2.2.1. Web Service cannot initiate communication with Client

In Web Services architecture, the Client can initiate communication with the Web Service but the Web Service cannot do the same. The Web service is never aware of its clients and cannot thus communicate its status to the clients, lets say if it was to go offline for maintenance.

2.2.2. Proxy is rewritten with every updating of the Web Service

Another limitation, but this time on the client side, is that every time the connection to the Web Service is refreshed, the client proxy has to be rewritten. This tedious process has been automated by leading vendors of Web Services development platforms. The downside to this advancement is that it will overwrite any custom proxy, which will be required if the client has to be configured to replace its Web Service dynamically.

These limitations had to be considered in the design of our solutions as you will see in the next subsection.

2.3. Approach for Solutions

2.3.1. Need of a custom client proxy that cannot be overwritten

The reconfiguration could be handled either from the Web Service side or the client side. In the case of a Web Service, it cannot initiate communication with the clients, to inform them of a switch because of the reasons explained in section 2.2. Web Service can only really be propped by hardware that can redirect client requests to another location, which allows for a limited scale of dynamic reconfiguration as mentioned in section 1.1.

On the other hand, if the Web Service is to be switched from the client side, the client can be manually configured to switch to a supplied URL on a detectable signal. This will require a custom client proxy and here some way will have to devised to prevent overwriting of this proxy by auto generated code, a problem explained in subsection 2.2.

2.3.2. External entity (Manager) to initialise Web Service switch

Some administrator controlled application has to signal the client proxy or the Web Service to initiate a web service switch and supply the client proxy with the new service URL. This application cannot be the Web Service because it cannot initiate conversation with the client, nor can it be the client itself, since we do not want the client application to initiate a Web Service switch on its own. Thus an external entity is required.

2.4. Dynamic Reconfiguration

The effect of dynamic reconfiguration on the original system needs to be measured to determine the effectiveness of the Reconfiguration. This led us to investigate the criterion for judging dynamic configuration. The results were:-

Correctness

- How the system is moved into a safe state before being reconfigured, such that no data is lost.

Efficiency

- Performance – how quickly an algorithm carries out change and how much disturbance is caused to the underlying system. Disturbance is the amount of deviation from the normal operation of the system.
- Measurement of how long a component is unable to receive data as a result of the reconfiguration process [5].

Transparency

- To what extent the programmers have to be aware of the presence of this system.

2.5. Investigated Technologies

2.5.1. Programming language and development platform

To start off, we had to determine which platform we would choose to develop our implementation in. The frameworks widely used in the current market to develop Web Services are Sun's J2EE and Microsoft' Dot Net. J2EE is a Java/JSP based technology and much older than Dot Net. Yet, we chose Dot Net C# for development because of following reasons. Firstly, while J2EE can support applications written in Java, Dot Net can support applications written in several languages including C#, C++, VB.Net and others. Secondly, though Java and J2EE are well tested and established products, they are modifying themselves to adapt to the Web Services architecture. Dot Net on the other hand has been specifically designed to cater for development based on Web Services architecture. Thirdly, due to Dot Net's Web Services Architecture focus, their development environment called Visual Studio .Net has been optimised for developing, testing and deploying Web Services. This means greater acceptance of Dot Net in the market for Web Services development, and our focus is on supplying the best solution for the most people [10].

2.5.2. Custom Client Proxy that cannot be overwritten

Here we had the option of modifying the auto-generated proxy, having a dynamic proxy or having a real proxy. These terms are explained within this section. Modification to Auto generated proxy was discussed in section 2.2.2. Since it gets overwritten every time the Web reference is refreshed, it was ruled out early on. Based on reasons outlined below we chose Real Proxy for implementation purposes.

The Dynamic proxy would have to be placed in between the auto generated proxy and the client code, such that the manager operations can be implemented prior to getting to the actual Web Service communication. All this functionality is used in the invocation handler method, which is the only method that gets run when the client makes a call to the Web Service (Dynamic Proxy instance). Here the dynamic proxy implements whatever it wants before passing the control to the actual method in the auto- generated proxy. Two explicit instantiations are made in this case, that of the Dynamic Proxy in the client code and that of the auto generated proxy in the Dynamic Proxy code. Note that the interface file for the auto-generated proxy is required, for Dynamic proxy generation [6].

The Real Proxy on the other hand is different from Dynamic proxy because it requires instantiation after the instantiation of the auto-generated proxy. The

real proxy wraps the auto-generated proxy in much the same way as dynamic proxy but it does not need an interface file, it only requires specification of the message transfer protocol. Thus their operation and their performance characteristics are very similar but transparency wise real Proxy is superior because it does not require an interface to be defined and adhered to [7].

2.5.3. Sockets Vs Remoting for external entity's means of communication

As discussed in section 2.3.2, an External entity (Manager) initializes the Web Service switch. The aim of investigating this subsection was to find the best means to send a simple signal to indicate a Web Service switch from the manager to either the Client or the Web Service and some text signifying the new address to switch to. We chose the Socket technique. It was because of the following reasons:-

Remoting requires a proxy/stub to be generated
A stub (proxy) is generated on the client side [10] in Remoting. This involves time and memory overhead and increases the disturbance in the Dynamic Reconfiguration process [7].

Sockets are the building blocks of Remoting but more importantly a separate proxy/stub is not required to setup and operate a new socket connection.

For socket communication, the client specifies the IP address and the port number of the Web Service it wants to talk to. Assuming the Web Service is actively listening for clients on that address, the connection is made. Communication can now take place both synchronously and asynchronously [8].

2.5.4. UDDI

UDDI stands for Universal Discovery Description and Integration. As mentioned in section 2.1, it is a registry of Web Service addresses and can be used to find information on Web Services. The point to note is that it can be used at runtime to publish and discover Web Service addresses. This could be potentially very useful in our project, since a Web Service Address can be loaded onto the UDDI and retrieved from it at runtime. In this case, the Manager will not be required to manually feed the client with the new Web Service Address on reconfiguration. The issue now was how we could identify a Web Service in UDDI uniquely, so as to use this capability. We found after much research, A Web Service can be identified using the keys below: -

- *Provider key* – identifies the Web Service provider organization or individual and can yield all the web services published from that provider.
- *Tmodel key* – identifies all web services, which follow a common WSDL interface (common public methods).
- *Service key* – identifies all services serving the same functionality but not necessarily following the same WSDL interface or having the same provider. E.g. services providing credit card facilities [9].

3. System Overview

We came up with two designs that dynamically reconfigured the Web Service. These designs were the Exception based design and the Interrupt based design. Both designs have been implemented in C# and successfully dynamically reconfigure Web Services at runtime by switching the Web Service to be maintained. This is done when the "maintenance administrator" controlled manager component (section 2.3.2 for details) initiates a switch between Web Services, while the system is in operation. The primary difference between the two designs is the nature of the manager. In the Exception based design, the manager is just another client of the Web Service, but with special access to it, whereas in the Socket design, the manager is a logically separate application, which explicitly communicates with the clients of the Web Services through sockets..

3.1. Exception Based Design

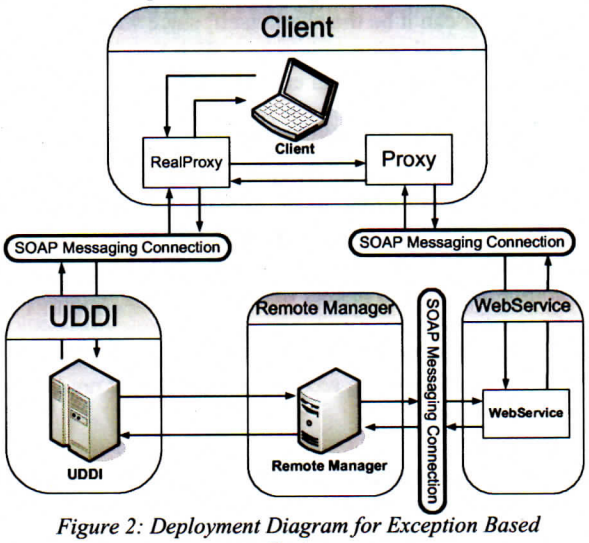


Figure 2: Deployment Diagram for Exception Based Design

3.1.1. Concepts

UDDI standards defined by us for Exception based design
We have defined a standard whereby each Web Service that can be switched maintains its address in the UDDI under a Service name and key. To avoid confusion, we

also defined in our standard that there shall only ever be one Web Service address listed under this Service key. Thus, if a Web Service has to be updated, the Manager Application can search for its parent service key in the UDDI and then replace the existing address with the new Web Service address. It is assumed that the user of the manager application is aware of and provides the service key of the Web Service to be updated and the new Web Service address.

Manager Application as a privileged client
The manager application connects to the Web Service like any other client of the Web Service. The only difference is that it has access to a privileged Web Method named, "`isBlocked(String Username, String Password)`"on the Web Service. This method tells the Web Service to block itself, such that all subsequent requests to the Web Service are responded to with a specific exception. The client's real proxy was programmed to catch this particular exception and then reconfigure itself. As is evident, we do not want any client of this Web Service to have access to this method, so we inserted the username and password parameters to limit access to this method.

3.1.2. Reconfiguration operation

The design operates in the following way: -

STEP 1
The Manager application accesses the UDDI and changes the Web Service Address under the Service key.

STEP 2
The Manager in its role of the privileged client, tells the Web Service to block itself. The Web Service acts accordingly and begins to respond to all further requests with an exception.

STEP 3
This step involves the client reactions to the exception generation of the Web Service in the previous step. The exception is picked up by the real proxies of the Web Service's clients when they make a request. The exception triggers them to automatically probe the UDDI and retrieve the new Web Service address specified by the Manager Application in step 1.

STEP 4
The client real proxies finally switch the Web Service address in their respective proxies to point to the new Web Service. All proceeding requests are then sent to the new Web Service.

3.2. Interrupt Based Design

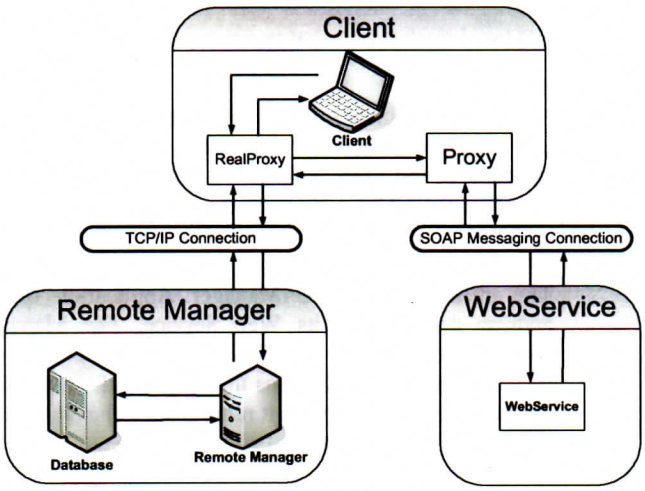


Figure 3 Deployment Diagram for Interrupt Based Design

3.2.1. Concepts

Manager aware of all connected clients
Every time a client connects to the Web Service, the client proxy automatically creates a socket connection to the manager application and sends it, its own address and its Web Service address. The manager application stores the client's address with its corresponding Web Service address, in its database of all active clients. We chose to keep this model, so we could easily search for the clients when the Web Service is to be switched and tell them all to change their Web Service addresses.

Manager communicating with all active clients through sockets
Sockets were used to connect up all the clients with the manager. They were used because of their light weight nature and because they do not require a stub/proxy as mentioned in section 2.5.3.

Manger handling starting up clients through database
Clients, which log in after their Web Service has been switched for maintenance, somehow needed to be informed of the new Web Service address. To handle this, we setup a standard whereby whenever a Web Service was switched, the address of the old Web Service (one that is switched) and the new Web Service (one that is switched to) was recorded in a table in the database. Thus, when a client connects up and provides the Manager with its Web Service address, the manager checks its database to see if it that Web service been switched. If so, then it informs the client of the new Web Service address.

3.2.2. Operation

The design operates in the following way: -

STEP 1
Every client establishes two connections when starting up. One connection to the Web Service as is the standard in the Web Services Architecture, and one to the Manager, which is done by the Real Proxy.

STEP 2
The client informs the Manager Application of its address and that of its Web Service. The Manager checks its database to see if that Web Service has been taken down. If so, it informs the client of the new Web Service address, otherwise it returns the same address as it was sent by the client.

STEP 3
This step and the ones following describe what happens when the user of the manager application decides to initiate a Web Service switch. The first step in this process requires the user of the manager application to select the Web Service to be taken down and the replacement Web Service from the user interface provided with the manager. Then give the command to switch.

STEP 4
This makes the manager to gather addresses of all clients registered with that Web Service from its database. The manager then sends all these clients the new Web Service address, as specified by the user in the previous step.

STEP 5
On receiving the new Web Service address, the client real proxies tell their respective proxies to switch to the new Web Service address. The way the Web Service architecture is implemented in the Dot Net framework ensures that the responses from previous requests and all requests after the Web Service switch has occurred are correctly handled.

NOTE: All communication between client and Manager is asynchronous for performance reasons.

4. System Evaluation / Comparisons

After having developed the implementations to our designs, we began to test and evaluate them. At this stage we referred back to the Reconfiguration evaluation criterion we developed in section 2.4 and others.

4.1. Correctness

Both designs complete the existing atomic request-response message interaction before switching, because calls to Web Methods in the Web Services Architecture are all synchronous, which we found out through testing. This ensures correctness in the dynamic reconfiguration process

4.2. Transparency

Transparency defines the extent to which the programmers of the Client and Web Service system have to be aware of the presence of the changes made by us (our frameworks) to make their system Dynamically Reconfigurable. This adds complexity to their development efforts. The table below shows the effects on transparency of the two designs.

Dynamically Reconfigurable Designs	Effect on Client side	Effect on Web Service side
Interrupt Based Design	The clients have to include the Real Proxy file. One line addition.	The clients have to include the Real Proxy file. One line addition.
Exception based design	none	The Web Service developer have to be aware of the `isBlocked()` method. This is because they have to include it in every Web Method of their Web Service implementations.

4.3. Closure

This refers to the ability of the Web Service to go offline when it has been switched. This would be relevant if the server on which the Web Service is running has to be taken down for maintenance. Interrupt based design is better in this regard.
In the Exception Based Design, even long after the Web Service has been switched, it cannot go offline. This is because the Web Service can never be sure that it has told its every possible client to switch, since the client has to communicate with the old Web Service prior to being informed of the switch, through an exception. On the other hand, in the Interrupt based Design, the manager handles all starting up clients, telling them to switch if their Web Service has been switched. This allows the web service using the Interrupt based design to go offline, if required for direct maintenance.

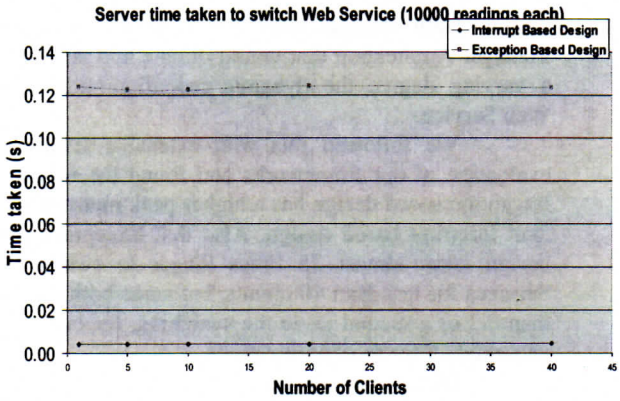
4.4. Efficiency

We quantified efficiency as:-

1. The time taken by the manager to initiate Web Service switching.
2. The amount of extra time taken in performing the standard tasks of Client Web Service interaction, due to the addition of dynamic reconfiguration functionality.
3. The increase in the amount of memory usage due to the addition of the dynamic reconfiguration functionality.

Our results were as follows: -

Time for manager to initiate a Web Service switch



In this test the time being measured in the two designs is: -

- Interrupt based design - The time between the Manager Application:
  - receiving the appropriate input from the user
  - to the time when the manager has informed all the clients of the Web Service of the new Web Service address.

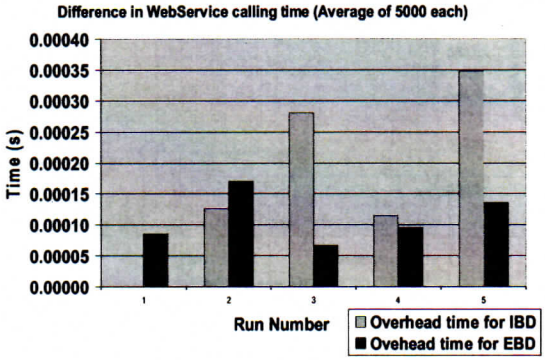
- Exception based design - The time for the Manager Application to execute the isBlock() Web Method on the Web Service.

We could not test the switching time for more than 40 clients because the computers in the software labs were not equipped for such computation. We found that both the designs took less than 0.2 seconds to perform the Web Service Switch on software lab computers for a maximum of 40 clients.

This time is too minimal to be noticed by the clients of the Web Service. We found that the Exception based design was a lot slower than the Interrupt based design. We noticed that with more clients the interrupt based design's time for Web Service switching increased whereas the Exception Based Design had a near constant time all throughout. This was expected because the Exception based design does not need to keep a record of or inform each active client of the Web Service to switch their Web Service.

Time lag due to addition of Our Frameworks

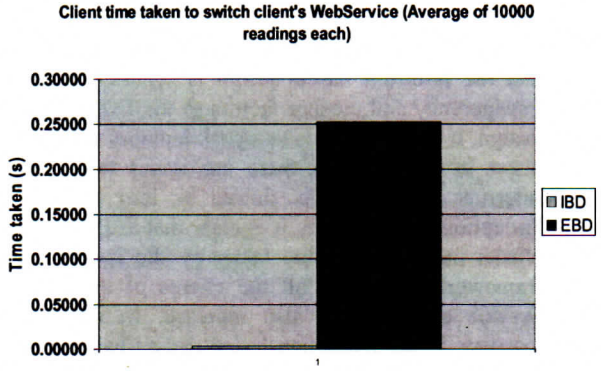
- Client Side: Web Service calling time



This test measured the time to call the Web Service from the client side. Thus, this time included the time to initialize the Real proxy and proxy for the both the designs and the associated marshalling and de-marshalling times.

The results clearly showed that the Interrupt based design took longer. This is because it has to make a socket connection to the Manager, on connecting to the Web Service; whereas the Exception based design only ever connects to the UDDI when it is asked to switch its Web Service.

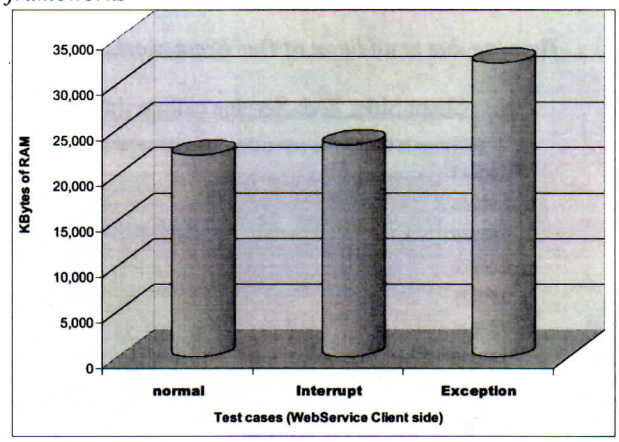
- Client Side: Web Service Switching time



This test measured the time to switch the Web Service on the Client side. Both designs need the Real Proxy to tell the Proxy to switch its Web Service address. But in the case of the Exception based design, the Real proxy has to itself connect to the UDDI and retrieve the Web Service Address. Thus the Exception

based Framework takes longer to switch the Web Service on the Client side as apposed to the interrupt based framework, as shown in the graph above.

Affect on Peak Memory Usage due to addition of our frameworks



The graph above compares the peak memory usage in normal Web Service operation with the Interrupt based Web Service operation and the Exception based Web Service operation. The test involved starting the client applications multiple times and recording their memory usage from the task manager. Then averaging the memory usage and plotting the graph.

Clearly the Exception based Design required a lot more in terms of memory than Interrupt and normal operation. This could be because the Real Proxy of the Client in the Exception based scenario maintains an app.config file, much like a text file with information on searching the UDDI.

5. Evaluation Discussion

It is clear from the Evaluation section above that the Interrupt based design is superior in-terms of transparency and closure factors to the Exception based design. It also consumes less peak memory and performs faster in general. Yet there are issues that make the Interrupt based design slower or less reliable than Exception based design. It is clear that as the number of clients increase the time taken by the interrupt based framework to inform all the clients of the new Web Service address will also increase. In addition, the manager in the Exception based design has to maintain socket connections over the network continuously for the period the client is active. This leaves too much faith on the reliability of the network, which is not guaranteed. Security is the next issue given there are so many live socket connections.

Thus it can be deduced that Interrupt based framework is better used in situations where there are a smaller number of clients and response times expected are short. An example of such a use could be in an online gaming website.

Exception based framework on the other hand is better used in situations where reliability is the key requirement and the response times can be of reasonable size. Example of this would be a Online banking website.

6. Conclusions

As stated in our objectives, we designed and implemented frameworks that allowed dynamic reconfiguration of Web Services in Microsoft DOT NET framework using the C# language. These designs were called Interrupt based design and Exception based design. Both designs used the idea of an external Manager Application that would initiate and facilitate to a varying degree the dynamic reconfiguration of the Web Service.

We followed this with extensive testing and evaluation of our frameworks and found the following. Exception based design has a higher peak memory usage than Interrupt based design. Also that Exception based design takes almost 25 times longer to switch Web Services for less than 40 clients, but since both take less than 0.2 of a second to do the switching, the lag will be insignificant for most applications.

Dynamic reconfiguration criteria of correctness and transparency were discussed in the System Evaluation/Comparison section among other criterion. Correctness was ensured by the synchronous and atomic request-response message interaction in the Web Service architecture. From the transparency point of view, we found Interrupt based design better because it did not introduce any changes to the Web Service side, when implementing dynamic reconfiguration.

In the discussion area, we discussed that Interrupt design was better suited for applications with fewer clients but an expectation of faster response times, whereas Exception based design was better suited for applications that require reliability over performance.

Finally, I am glad to have been part of this project and having learnt so much about team work and the field of distributed networks. If I were to do the project again, I would keep more regular records of our team meetings, which help a lot when writing reports.

Acknowledgements

The student would like to thank the supervisor Mr. Ian Warren, second supervisor Mr. Colin Coghill and of-course the eight month partner of this project and good friend, Gene Lee. Also I would like to thank Alan Chan and Oliver Hunter for their support throughout the project period.

7. References

[1] Web Service, *Wikipedia*, accessed on 01 September, 2005 from http://en.wikipedia.org/wiki/Web_services

[2] Warren, I. and Hillman, J., "An Open Framework for Dynamic Reconfiguration", IEEE Computer Society: 26th International Conference on Software Engineering (ICSE'04), pp: 594-603, 2004

[3] Williams, J., Avoiding the CNN moment, IT PRO (an IEEE publication), March-April 2001, Page 72

[4] Loney, M., The state of Web services http://insight.zdnet.co.uk/internet/webservices/0,39020460,2137913-1,00.htm

[5] Warren, I. and Hillman, J., "Quantitative Analysis of Dynamic Reconfiguration Algorithms", presented at the International Conference on Design, Analysis and Simulation of Distributed Systems in Virginia, USA, 2004.

[6] SKI_BUM, Dynamic/Transparent Proxy using Dynamic Proxy.Net, The Code Project [Available] http://www.codeproject.com/dotnet/dynamicproxy.asp

[7] Agrawal, A. K., "Distributed Computing using . Net Remoting", C# corner [Available] http://www.c-sharpcorner.com/Code/2004/Jan/NetRemoting.asp

[8] Dhar, Ashish, Socket Programming in C# - Part 1, [Available] http://www.developerfusion.com/scripts/print.aspx?id=3918

[9] Lee, G. and Sarna, J., *Project Management Summaries*

[10] Rao, Nageshwar T.V; XML-Based Web Services in J2EE and .Net, [Available] http://students.cs.tamu.edu/jchen/cpsc689-608/comparison/nrt3364-comparison_nagesh.pdf