Web Services Loan Application

UCL

Birkbeck College

**Department of Computer Science and Information Systems**

Final year project undertaken in partial fulfilment of the requirements for the

BSc (Honours) Degree in *Information Systems and Computing*

Web Services Loan Application

By

Mahfoud Deboub

Supervisor: Professor George Magoulas

Project Unit: (COIY032S6)

2012

**Abstract**

Many applications have been written in different languages and can in some cases run on specific platforms. Sometimes these applications need to talk to each other, which can be a very hard and expensive task to achieve.

Web Services can help to solve the problem of communication between applications. With Web Services applications integration can be achieved easily, because Web Services provide a common form for application communication regardless of the platform or the implementation language.

The aim of this paper is to produce a Loan Comparison Application, to expose and discuss the main building blocks of a Web Service technology. The Loan Comparison application is a simple E-commerce application made up of several components. The functionality of the Loan Comparison system is kept to minimum, the reason behind this is to concentrate and focus mainly on the development and the implementation of Web Services.

The loan comparison application offers two services to its client: the first service is a loan comparison service, which displays to the client the best loan offered from different banks (Lenders). The second service offers a credit rating to client based on his /her employment status, salary and National Insurance Number.

Table of Contents

[Chapter 1 - Introduction 1](#_Toc323826841)

[1.1 Introduction 1](#_Toc323826842)

[*1.2* Aims and objectives 2](#_Toc323826843)

[1.3 Project Plan 3](#_Toc323826844)

[1.4 Summary 4](#_Toc323826845)

[Chapter 2 -Review technologies 5](#_Toc323826846)

[2.1 Problems with other Technologies 5](#_Toc323826847)

[2.2 CORBA 5](#_Toc323826848)

[2.3 DCOM 7](#_Toc323826849)

[2.4 Java/RMI 8](#_Toc323826850)

[2.5 The Web Services Solution 10](#_Toc323826851)

[2.6 Benefits of Web Services 11](#_Toc323826852)

[2.7 Summary 12](#_Toc323826853)

[Chapter 3 – Requirements 13](#_Toc323826854)

[3.1 Research 13](#_Toc323826855)

[3.2 Requirements capture 15](#_Toc323826856)

[3.3 Summary 16](#_Toc323826857)

[Chapter 4 –Methodology 17](#_Toc323826858)

[4.1. Introduction to Software Development Methodology 17](#_Toc323826859)

[4.2 The project Methodology 17](#_Toc323826860)

[4.3 Web Services 18](#_Toc323826861)

[4.3.1 The underline technology of Web Services: 19](#_Toc323826862)

[4.3.2 Web Services Architecture 20](#_Toc323826863)

[4.3.3 WSDL (Web Service Description Language) 21](#_Toc323826864)

[4.3.4 UDDI (Universal Description, Discovery and Integration) 22](#_Toc323826865)

[4.4 Summary 23](#_Toc323826866)

[Chapter 5 - Designs and Analysis 24](#_Toc323826867)

[In this chapter the Loan Comparison Application System Architecture will be developed, and all the system components will be designed according to Object Oriented (OO) approach. 24](#_Toc323826868)

[5.1 System Analysis 24](#_Toc323826869)

[5.2 Application Components: 24](#_Toc323826870)

[5.3 An overview of the System Design 25](#_Toc323826871)

[5.4 CompareLoanWS 26](#_Toc323826872)

[5.5 CreditRating 29](#_Toc323826873)

[5.6 Databases 32](#_Toc323826874)

[5.7 User Interface 34](#_Toc323826875)

[5.6 Summary 36](#_Toc323826876)

[Chapter 6 - Implementation 37](#_Toc323826877)

[6.1 Software and Tools Used 37](#_Toc323826878)

[6.1.1 J2EE 37](#_Toc323826879)

[6.1.2 WebLogic Server 10gR 37](#_Toc323826880)

[6.1.3 Workshop for WebLogic 38](#_Toc323826881)

[6.1.4 MySQL 38](#_Toc323826882)

[6.2 Implementing and Connecting Databases 38](#_Toc323826883)

[6.3 CompareLoanWS Web Service 40](#_Toc323826884)

[6.3.1 CompareLoanWS Class 40](#_Toc323826885)

[6.3.2 Loan Class 42](#_Toc323826886)

[6.3.3 LoanControl Interface 42](#_Toc323826887)

[6.3.4 CompareLoanWSServiceControl Interface 43](#_Toc323826888)

[6.4 CreditRating Web Service 44](#_Toc323826889)

[6.4.1 CreditRating Class 44](#_Toc323826890)

[6.4.2 CalcCreditRating Class 46](#_Toc323826891)

[6.4.3 BankruptcyControl Interface 47](#_Toc323826892)

[6.4.4 CreditRtingServiceControl Interface 47](#_Toc323826893)

[6.5 User Interface 49](#_Toc323826894)

[6.5.1 ClientPageFlowController 49](#_Toc323826895)

[6.5.2 JSPs (Java Server Pages) 51](#_Toc323826896)

[6.6 – Summary 54](#_Toc323826897)

[Chapter 7- Testing 55](#_Toc323826898)

[7.1 – CreditRating Web Service Testing 55](#_Toc323826899)

[7.2 – CompareLoanWS Web Service Testing 55](#_Toc323826900)

[7.3 – User Testing 58](#_Toc323826901)

[Chapter 8 - Conclusion 59](#_Toc323826902)

[Chapter 7 - Appendix 60](#_Toc323826903)

[Chapter 8 - REFERENCES 61](#_Toc323826904)

# Chapter 1 - Introduction

## 1.1 Introduction

What the Internet has been able to do, is provide companies with a very low cost advertising medium capable of reaching more people and provide a low cost trading service in any area.

Software to support the small business and their collection of users has not been widely produced, while there are plenty of software solutions for large firms.

Web Services is a new paradigm for distributed systems, which offers an exciting new model for creating robust, secure, extensible and versatile solutions. With its standards-based core technologies, Web Services provide the interoperability and ubiquity to make it the best solution technology for small and medium size businesses.

E-commerce is the trading of goods and services on the internet, with E-commerce businesses and users have reduced the cost of transactions as well as the time taken. There are two types of E-commerce; B2B (Business to Business) where businesses can exchange products and services and B2C (Business to Customer) like Amazon and eBay which is from business direct to customer.

Comparison websites enable users to specify their requirements in order to be presented with a range of offers from different providers. Both users and providers have benefited from comparison websites, by saving time and money. Many businesses like Financial Services, communication and Energy Services, are becoming more reliant on comparison websites to sell they products.

The business model of comparison websites varies from model to model. Most of the comparison websites don’t charge the users for using their services. Instead they make their profit from providers (Retailers), who advertise their products on the comparison website. Providers either pay a flat rate to the comparison website or are charged for each click by user on their link or for each time a sale is successful.

There are two main technologies used in comparison websites:

**Screen Scarping**

This technology was introduced when comparison websites first appeared. It involves copying the product or service information from the provider webpage to the comparison website for use by the user. This presents a lot of problems as HTML documents copied could not be translated in a meaningful way. In addition to this problem, as more providers offer their products more pages needed to be scraped.

**XML**

Another solution is that providers let comparison websites access their back-end data facilities using XML technology. The comparison websites will have to access the provider server in order to collect the data and information needed about the product offered.

## *1.2* Aims and objectives

The aim of this paper is to demonstrate the use of Web Services, and to present an initial investigation of the technology involved in the implementation of Web Services. To achieve this aim, a simple loan comparison application will be developed and analysed to explain all the stages required in developing and implementing a Web Service system.

* Explore the Web Services technology with the intention of developing Web Services based system.
* Develop a prototype.
* Achieve an extensive understanding of the Web Services model.
* Design a new system.
* Implement the design, resulting in a prototype artefact.
* Test and evaluate the artefact.
* Summarise the project and discuss recommendations for future work based on the prototype.

## 1.3 Project Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NO.** | **Task** | **Duration** | | **Date** |
| 1 | Project research (Web Services) | 15 days | | 15/10/2011 |
| 2 | Proposal | 7 days | | 07/11/2011 |
| 3 | Investigating the Web Services concept | 30 days | | 07/12/2011 |
| 4 | Searching Web Service Applications | 20 days | | 02/01/2012 |
| 5 | Design a simple Loan application | 18 days | | 20/01/2012 |
| 6 | Investigating WebLogic Server and its underlining technology | 30 days | | 27/02/2012 |
| 7 | Learning about Workshop for WebLogic Server | 14 days | | 10/02/2012 |
| 8 | Learning how to connect and access databases with MySQL and WebLogic Server | 10 days | | 20/02/2012 |
| 9 | Developing Web service applications with Oracle technologies | 10 days | | 02/03/2012 |
| 10 | Integrating the user interface with PageFlow Technology | 3 days | | 05/03/2012 |
| 11 | Testing the application | 3 days | | 08/03/2012 |
| 12 | Investigating security technologies in J2EE and web services | 7 days | | 15/03/2012 |
| 13 | Debugging the User Interface | 7 days | | 22/03/2012 |
| 14 | Documentation | 31 days | 22/04/2012 | |
| 15 | Bug Fix/Testing | 7 days | 30/04/2009 | |
| 16 | Completion of Project | 7 days | 07/05/2012 | |

## 1.4 Summary

In this chapter the aims and the objectives of the project were established with a quick introduction to Web Services, E-commerce and the technology behind them. Also the project plan and mile stones have been setup.

.

# Chapter 2 -Review technologies

This chapter will look at the different technologies that can be used in E-commerce comparison applications, followed by a comparison between these technologies. The chapter finishes by introducing Web Services and its benefits.

## 2.1 Problems with other Technologies

The Loan Comparison application is made up of several components or applications that needs to collaborate together to complete a certain task. Therefore the type of architecture needed to develop this application has to be based on a technology, that enables all the different components to communicate freely and allow the Loan Comparison application to expand without the need for changing its core structure.

Another issue that the Loan Comparison application needs to address, is the possibility that some of its components may be built with a different language or run on a different platform. Users nowadays access websites and organisations’ resources by using different devices (PDA, Mobile, etc.). One of the emerging technologies that can be used to construct this architecture is Middleware.

Middleware is piece of software that resides between the operating system and the application and allowsmultiple applications to connect; mostly in remote environment via a network. The technologies used in the implementation of middleware vary in architecture, design and communication.

Two of the leading technologies in distributed system are ***CORBA*** (Common Object Request Broker Architecture),and Microsoft ***DCOM*** (Distributed Component Object Model). Both these technologies are based upon RPC (Remote Procedure Call), in which a computer on a network can access a program that resides on another computer.

## 2.2 CORBA

CORBA is an object-oriented middleware which extends the procedural type of middleware. The remote invocation mechanism involved in CORBA allows the implementation of the object-oriented features such as exceptions, inheritance and object reference. CORBA is an architecture and specification for creating, distributing, and managing distributed program objects in a network.

CORBA is an acronym for Common Object Request Broker Architecture; the Object Request Broker (ORB) is a mechanism for calling methods or procedures on an object locally or remotely to provide a function or service for the caller (client). This object can represent something in the real world like a shopping cart, or for the Loan Comparison application, it can be a request for a service, the ORB implements location transparency. Exactly the same request mechanism is used by the client and the CORBA object regardless of where the object is located.

The services that an object provides are given by its interfaces as defined in Interface Definition Language IDL, which is programming language neutral. IDL defines language bindings for many different programming languages. This allows an object implementer to choose the appropriate programming language for the object.

The common aspect of CORBA is defined in its technical standard, which is set by a not-for-profit organisation called Object Management Group (OMG) that offers many benefits in CORBA; OMG was formed in 1989 to create specifications for open distributed computing.

The great popularity of the Internet and the complexity of modern applications meant that communication and data transactions over the internet, and the large local networks, have become more difficult to manage and process. Therefore interoperability becomes a primary requirement for organisations and businesses.

The CORBA architecture is based on the object model where objects are abstracts; object reference is used to identify a CORBA object. Clients use an object reference to invoke requests on a CORBA object, each CORBA object has a clearly-defined interface, specified in the CORBA Interface Definition Language (IDL). A client needs only to refer to the object’s interface definition.

First step in developing a CORBA application is defining interfaces to objects, using IDL then compiling the IDL file to generate code in Java or other languages. This code includes a client stub code which is developed by client programs, and an object skeleton code to implement CORBA objects.

A client invokes on object references that it obtains from the server process. The ORB then passes the function call through the object skeleton code to the target object by Generating a unique ID called CORBA Object Reference for the server, finally ORB can activate or deactivate the server as required.

Figure 2.1 provides an overview of CORBA system.

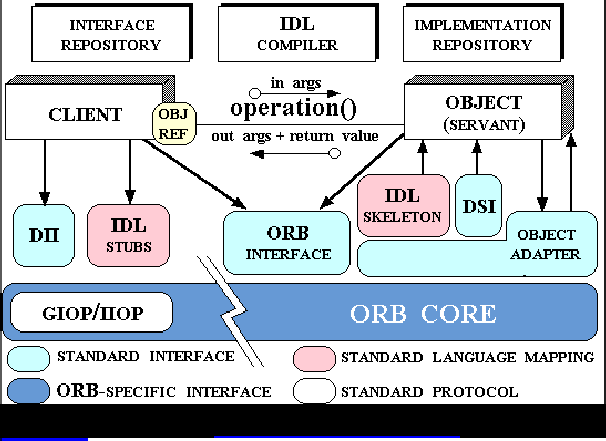


Figure 2.1 CORBA Overview <http://www.cs.wustl.edu/~schmidt/corba-overview.html>

**Disadvantages of CORBA**

* Limitations in passing through Firewalls.
* Very complicated to implement.
* No reference implementation.
* Problem with scalability due to tied coupling.

## 2.3 DCOM

Microsoft's Distributed Component Object Model (DCOM) is run on a protocol called Object Remote Procedure Call (ORPC).

“DCOM is based on the original Distributed Computing Environment (DCE) standard Remote Procedure Call (RPC) infrastructure and was created as an extension of the Component Object Model (COM) to allow the creation of server objects on remote machines. In order for COM to create remote objects, the COM libraries need to know the network name of the server. Once the server name and CLSID (a globally unique identifier representing a COM Class within the server) are known, the Service Control Manager (SCM) on the client machine connects to the SCM on the server machine and requests creation of the remote machine's server object. Because DCOM is an extension of COM, it relies on the registry and COM libraries to supply the type library information of the object to create on the remote server machine. The remote server name is either configured in the registry or passed as an explicit parameter to a CoCreateInstanceEx call (in Visual Basic, this would be a CreateObject call).” [9]

Figure 2.2 provides an overview of DCOM system.

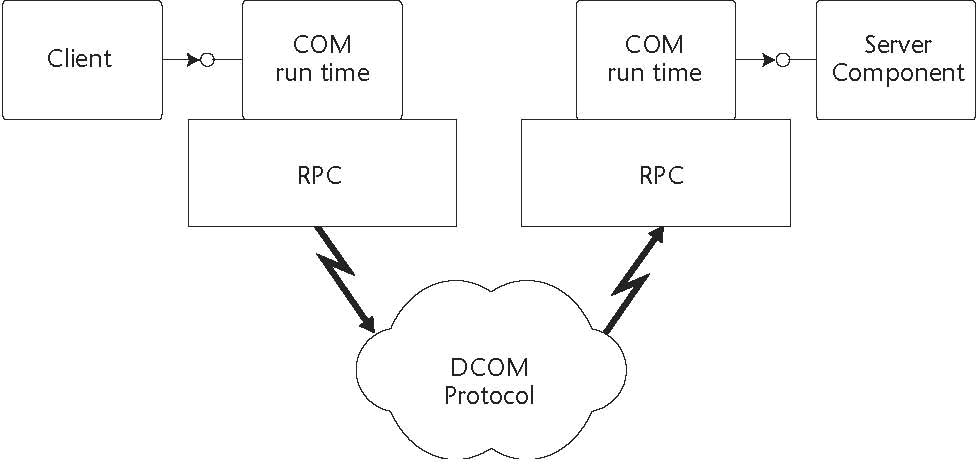


Figure 2.2 DCOM Overview <http://web.info.uvt.ro/~petcu/distrib/TDS1.pdf>

**Disadvantages of DCOM**

* Microsoft application environments only.
* Problem with scalability due to tied coupling

## 2.4 Java/RMI

Java/Remote Method Invocation uses a protocol called Java Remote Method Protocol (JRMP). The object marshalled (transmitted) in RMI has to be written in Java for both the client and server. Unlike other distributed programming interfaces (RPC, IDL, etc.), Java RMI is language specific. This means that more advance features can be introduced, rather than trying to meet the lowest common ground.

“The first layer is the Stub/Skeleton Layer. This layer is responsible for managing the remote object interface between the client and server. The second layer is the Remote Reference Layer (RRL). This layer is responsible for managing the "liveliness" of the remote objects. It also manages the communication between the client/server and virtual machine s, (e.g., threading, garbage collection, etc.) for remote objects.

The third layer is the transport layer. This is the actual network/communication layer that is used to send the information between the client and server over the wire. It is currently TCP/IP based. If you are familiar with RPC, it is a UDP-based protocol which is fast but is stateless and can lose packets. TCP is a higher-level protocol that manages state and error correction automatically, but it is correspondingly slower than UDP.” [10]

Figure 2.3 provides an overview of the Java/RMI system.

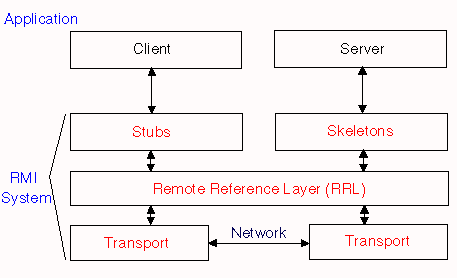


Figure 2.3 Java/RMI Overview <http://www.edm2.com/0601/rmi1.html>

**Disadvantages of Java/RMI**

* RMI is limited to the Java platform.
* RMI architecture is tightly coupled.

Figure 2.4 Overview of Architecture comparisons of middleware systems.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***CORBA*** | *Common Data Representation (CDR)* | *Interoperable Object Reference (IOR)* | *IIOP (Binary)* | *Interface Definition Language (IDL)* |
| **DCOM** | Network Data Representation (NDR) | OBJREF | DCOM (Binary) | Inherited from COM |
| **RMI** | Serializable Java Objects | URL | Java Remote Method Protocol (JRMP) | Java Interfaces |
| **Web Services** | XML | URL | SOAP (Text-based) | WSDL |

Figure 2.4 Middleware Systems

## 2.5 The Web Services Solution

CORBA, DCOM and Java/RMI technologies comes short as the preferred choice of architecture to implement the Loan Comparison application. Those technologies are very useful when it comes to integrating applications within a homogenous environment inside a LAN. Applications implementing different protocols on heterogeneous platforms become very hard to achieve.

In the Internet environment a different approach is needed to address the interoperability of applications and to solve the application -to -application communication over the network problem. This approach is required to be:

* *Platform and language independent.*
* Open.
* Easy to implement.
* Standardised.
* *Extensible.*

Web Services technology meets all the above requirements, and can be just the solution for the Loan Comparison application. Applications integration becomes very simple to achieve, because Web services is platform and language independent. Therefore application interior implementation (Business logic) can be changed, without modifying its interface.

Web Services supports many types of technologies like B2B, ED, etc. Also web services are self-describing software modules, which make them the first choice for Loosely Coupled application integration.

## 2.6 Benefits of Web Services

Web Services has many benefits to application integration, for example:

* **Standards-Based**

Web Service uses XML, HTTP and SOAP protocols, these protocols are the standard in internet commutation nowadays. These standards are common for any platform.

* **Reusability**

With Web Services legacy applications can be wrapped to communicate freely with other applications. Functions used in Web Services can be moved and reused time and time again in other applications.

* **Interoperability**

Web Services supports many clients regardless of the platform or the language they use to communicate with it. Therefore Web Services are platform and language independent.

* **Flexibility**

XML documents can be changed over time to incorporate more functionality of the Web Service. This flexibility allows the application to extend indefinitely.

* **Reliability and Scalability**

Web Services are built and compiled to J2EE standard specification, which requires reliability and scalability in developing enterprise applications.

* **Cost savings**

Over the years organisations has spent vast amounts of money to maintain and develop applications, with the introduction of Web Services applications can be built on the fly with minimal cost and lesser staff.

Also legacy applications integration can be very costly and can consume lot of resources. With Web Services legacy applications can be wrapped and integrated with other applications, without the need for further development.

## 2.7 Summary

This chapter explained in depth the different Middleware technologies used to develop E-commerce comparison applications. These technologies include CORBA, DCOM AND Java/RMI; the chapter also mentioned the advantages and disadvantages of each one of these technologies regarding E-commerce.

The chapter finished by covering the Web Services technology and its benefits and why this technology was preferred to the others for Loan Comparison Application Development.

# Chapter 3 – Requirements

This chapter will look in to the project application Loan Comparison Application requirements phase, including functional requirements.

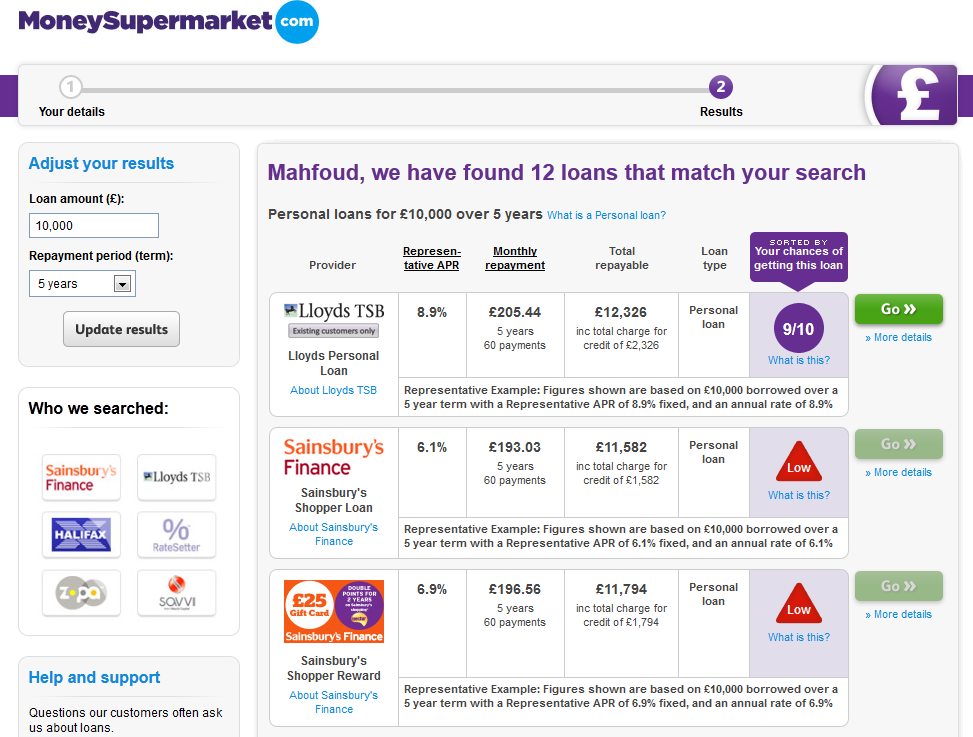
## 3.1 Research

System requirements are based on simulation scenarios to investigate and demonstrate the use of Web Services Technology in a Comparison Application.

A thorough research was conducted mainly on the Internet about similar scenarios and applications, that share some similarity to the system that will be developed in this project. Here are some samples of similar applications and their core functionality:

* **MoneySupermarket.com**

After the user submits some imformation regarding the loan, the user is presented with a set of quotes for a loan from different loan providers.



* **Comparethemarket.com**

Here the user experience is very simple and not time consuming. The user selects the amount of money he/she want to borrow and the period of the loan. Then the best loan quotes are displayed instantly.

****

* **Oracle Documentation**

[**http://docs.oracle.com/cd/E13226\_01/workshop/docs70/help/index.html#guide/tutorial/tutFirstWebServiceStep1.html**](http://docs.oracle.com/cd/E13226_01/workshop/docs70/help/index.html#guide/tutorial/tutFirstWebServiceStep1.html)

By searching the Investigate tutorial from the Oracle tutorials, and using the WebLogic server, the Loan Comparison application development will be based on Oracle technology.

There are lot of websites similar to those above, which offer loan comparison services. The best site tends to keep user input and interaction with the service to the minimum, in other words to do exactly what it is meant to do.

## 3.2 Requirements capture

The requirements gathered for the system are:

1. The Loan Comparison application receives a request from a client, regarding Loan deals
2. The Loan Comparison Application checks a Database where all the loan deals from different banks (lenders) are stored.
3. The Loan Comparison application responds to the client (applicant) with the loan deals results.
4. The Loan Comparison application offers another service, in which it receives a request from a client, regarding a credit rating check.
5. The Loan Comparison Application requests a credit check from Credit Rating Agency to get client credit rating. (External Service)
6. The Credit Rating Agency application checks its own Database for bankrupt clients. (Internal Service)
7. The Credit Rating Agency application calculates the applicant’s credit score by using an EJB. (Internal Service)
8. The Credit Rating Agency application responds to The Loan Comparison application with the user’s credit rating. (External Service)
9. The Loan Comparison application responds to the user (applicant) with the credit rating result.
10. The system needs to have a mechanism where to restrict client input to only the accepted type used by the service provided, otherwise an error message is displayed to the client.

## 3.3 Summary

This chapter started by giving some examples of a well-known loan comparison web sites and sample applications obtained from Oracle WebLogic Documentation. Then the System requirements were gathered for the design phase.

# Chapter 4 –Methodology

This chapter will discuss the different methodologies used in System Development and the preferred methodology for developing the Loan Comparison Application. The chapter will finish by talking about Web Service and its architecture.

## 4.1. Introduction to Software Development Methodology

“A methodology in the domain of IS must cover a number of aspects of the project, although coverage varies from one to another. A vision and Fitzgerald (2002) describe a methodology as a collection of many components. Typically, each methodology has procedures, techniques, tools and documentation.”-[7].

Methodology helps to produce better quality software by following a certain standard and mile stone. Each methodology is best suited to specific kinds of projects. A short description of some of these approaches is:

* **Waterfall or Traditional Lifecycle (TLC)** is a sequential approach in which each stage has been completed before moving to the next one. The stages are: Requirements analysis, Design, construction (Implementation), Testing, Installation and Maintenance. It suits systems development when the requirements are very clear.
* **Incremental** or Iterative is an approach in which the project development can be viewed as mini –project in its own right. “Gilb (1988) suggested that successful large systems start out as successful small systems that grow incrementally.” –[8]
* **Agile (Extreme programming)** one the keys features of this approach is that user requirements change over the lifecycle of the project.

## 4.2 The project Methodology

Based on the requirement of the project and the technology that is introduced (Web Service), the methodology chosen that suits the development of the Loan Comparison application is the Incremental approach. The reasons behind this decision are mainly to do with; 1) as mentioned in the requirements capture before, the lack of a real client and the nature of the project that consist of few functionalities. 2) Web Services systems are made up of loosely coupled systems. Therefore each entity can be regarded as a separate system in its own right, since having a completely functional product at the end is not necessarily the primary objective.

Figure4.1 shows the Incremental Approach

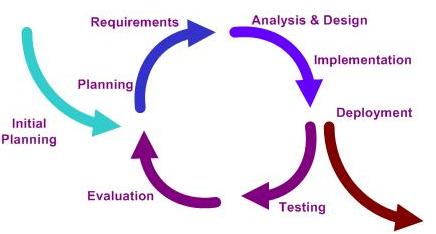
****

Figure 4.1 Incremental Approach

## 4.3 Web Services

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards” – [1].

The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services- [2].

Web Services are more suitable for *loosely coupled* systems, where the client might have no prior knowledge of the Web Service until it actually invokes it. These make Web Services the ideal technology when it comes to applications communication over the Internet.

### 4.3.1 The underline technology of Web Services:

* **Loosely coupled** applications can change business logic without the need for other applications that communicate with it to change anything about the way they invoke it.
* **Encapsulation** is one of the fundamental bases of OO (Object Oriented) approach. The service hides its interior implementation and exposes it only via an interface.
* **Open Standard Protocols** XML (Extensible Mark-up Language) and HTTP Hypertext Transfer Protocol and SOAP (Simple Object Access Protocol).
* **XML** Web Services uses a simple XML-based protocol to allow applications to exchange data across the Web. A client sends a XML message containing the request to the Web Service to do a particular operation. The Web Service replies with an XML message containing the response (result) of the requested message.

“XML syntax consists of text-based mark-up that describes the data being tagged, it is both application-independent and human readable.” – [6]

* **SOAP** is an XML based communication protocol for interacting with Web services. It specifies a format in which methods and data can be passed from an application to another. Therefore web services can be available to any client application capable of sending and receiving the appropriate XML messages. SOAP messages have a general structure:

1) SOAP Envelope – defines the content of the message.

2) SOAP Header – (Optional component) contains header information and extensions. Security contexts are usually placed here [WL33].

3) SOAP Body – Contains call and response information, including the payload and SOAP Fault catching mechanism.

Figure 4.2 provides an overview of a SOAP message.

****

Figure 4.2 SOAP message

* **HTTP** To make a Web Service accessible to other applications across networks, such as the Internet and in−house intranets, web

Services receive requests and send responses using widely used protocols such as Hypertext Transfer Protocol (HTTP) and Java Message Service (JMS). One of the major benefits of HTTP is its ability to go through Firewalls.

### 4.3.2 Web Services Architecture

The main architecture of the Web Service is composed of provider, requester and register.

* Service Provider develops the service and publishes its functionality (interface) to the registry UDDI (Universal Description, Discovery and Integration) by creating a WSDL (Web Service Description Language) file, which describe what the service offers and how to invoke it.
* Service Requester is any client who invokes the service, first the client discovers the service through the UDDI or the registry then Binds to it by opening a connection channel.
* Service Registry or UDDI are centralised directories of services, where clients can look up for Web Services published by the Service Providers.

Figure 4.3 provides an overview of a Web Service Architecture

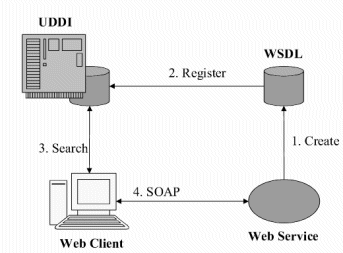


Figure 4.3 Web Services Architecture

### 4.3.3 WSDL (Web Service Description Language)

WSDL is a standard XML document type controlled by the World Wide Web Consortium (W3C, see [www.w3.org](http://www.w3.org/TR/wsdl) for more information). [3]

A Web Service describes its operations and how to interface with it through a Web Service Description Language (WSDL) file. Also it uses the WSDL file to specify which data type input it requires to invoke its operation and what return data type the client can expect.

The WSDL file contains:

* Data type for the Web Service methods and the type of the returned data.
* All the Web service methods (Operations).
* Methods communication formats and the protocols corresponding to it.
* The URL of the Web Service.

Figure4.5 shows WSDL document format

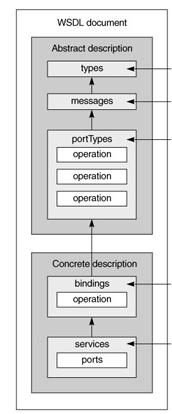
****

Figure 4.5 WSDL file format

### 4.3.4 UDDI (Universal Description, Discovery and Integration)

UDDI is supported cross-industry by major vendors (IBM, BEA, DELL, Oracle, and Microsoft ….ext.), Often described as the yellow pages of Web services.

“Welcome to [UDDI XML.org.](http://uddi.xml.org/) This is the official community gathering place and information resource for the   
[UDDI OASIS Standard](http://www.oasis-open.org/specs/index.php#uddiv3.0.2), which defines a universal method for enterprises to dynamically discover and invoke Web services. The standard is advanced through an open process by the [OASIS UDDI Specification Technical Committee](http://www.oasis-open.org/committees/uddi-spec), a group that encourages new participation from developers and users. This is a community-driven site, and the public is encouraged to [contribute content.](http://uddi.xml.org/contribute-content)”- [4]

“UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet.” - [5]

## 4.4 Summary

In this section the different methodology and approaches to system development have been addressed, and the particular approach chosen to develop this project application was discussed.

Also the Web Services technology was discussed in detail and all its major building blocks (WSDL, UDDI, SOAP and XML).

In the next section the design stage of the Loan Comparison Application will be discussed in detail, including the development of the System architecture and its complements.

# Chapter 5 - Designs and Analysis

## In this chapter the Loan Comparison Application System Architecture will be developed, and all the system components will be designed according to Object Oriented (OO) approach.

## 5.1 System Analysis

As discussed in previous chapters the aim of developing the Loan Comparison application is to expose the Web Services architecture, at the core of Web Services is the idea of Loosely- Coupled Services. Therefore when designing the Loan Comparison application, each component will be designed and developed as a separate entity or module.

This approach has many benefits:

* Any particular part of the application can be tested separately, as well as tested with other modules too.
* With the modular approach the MVC (Model, View and Controller) can be implemented.
* Other components can be added to the system in future.

## 5.2 Application Components:

* The loan comparison service checks the in- house database of loan details.
* The credit rating service invokes another Web Service (Credit Rating Agency).
* The Credit Rating Agency is a separate Web Service system, which in itself has a database and EJB (Enterprise Java Bean).
* The client interface is implemented using page flow to achieve an MVC (Model, View and Control) approach.

The application consists of three modules:

* Loan Comparison Web Service.
* Credit Rating Agency Web Service.
* Web Interface to gather user input and display output.

Figure 5.1 provides a conceptual overview of the system.

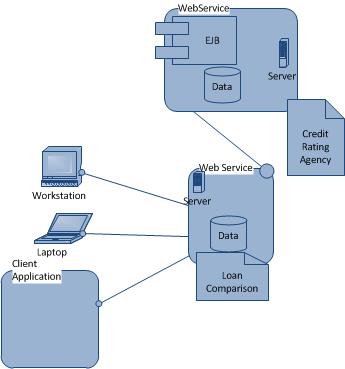


Figure 5.1Conceptual Overview of the System

## 5.3 An overview of the System Design

The Design of the Loan Comparison Application will be based on Oracle WebLogic Server Technology. At the heart of WebLogic is something called Controls, any connection between Web Services will be handled by Service Control, and also Database Control will manage the connection between Web Services and the databases.

The reason behind the choice of WebLogic Controls technology is the idea at the heart of Web Services architecture: loose coupling. The code of the Controls interfaces file is not implemented in the Web Service file, therefore the Controls’ interfaces can be changed, replaced or used for another Web Service.

Also for the same reason that applies for Controls, the use of EJB (Enterprise Java Beans) enforces the idea of Loose Coupling. This is one of the fundamentals of the OO (Object Oriented) approach.

More details and analysis of this technology will be discusses in the coming chapters.

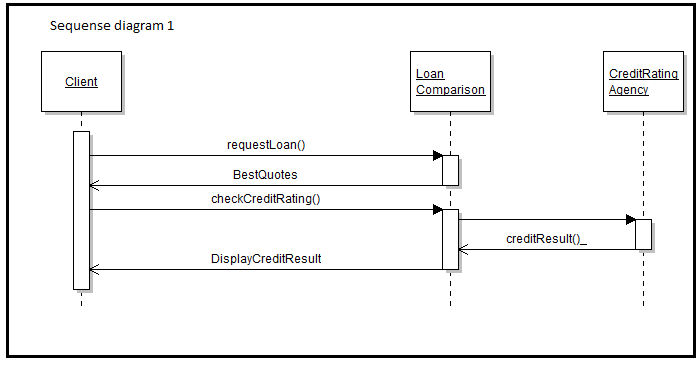
****

Figure 5.2 Sequence diagram showing an overview of the application.

## 5.4 CompareLoanWS

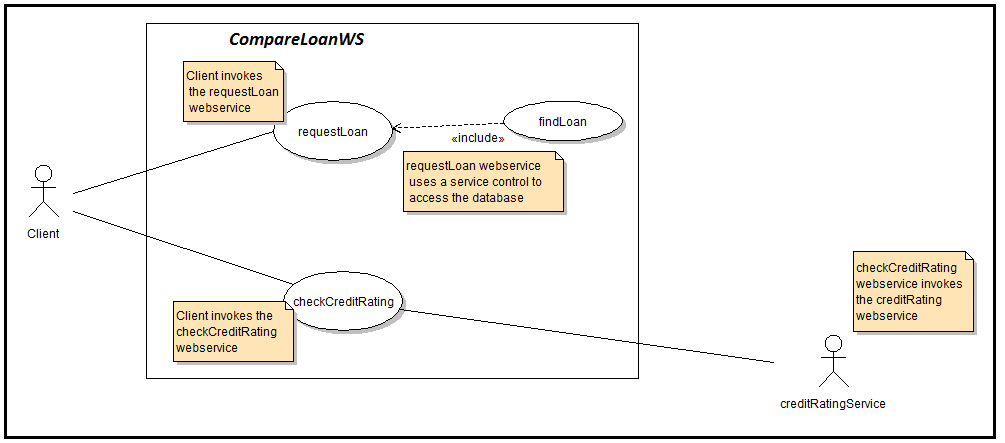


Figure 5.3 Use Case diagram of the CompareLoanWS System.

At the core of the Loan Comparison application is the Loan Comparison Web Service System. The figure 5.3 shows the use case diagram of this Web Service, which exposes two of its methods as services to client, and it contains an in-house database.

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| requestLoan | A user submits a form which contains some of his/her details, when the requestLoan method of the CompareLoanWS is invoked, this triggers the findLoan method of the **Database Control**, which checks the databases and sends a reply to the user with a list of loan deals from providers. |
| findLoan | When a requestLoan method is invoked, the findLoan method, which is a **Database Control method**, is invoked too. By selecting from a database list of the best loan deals, then returning the result to the requestLoan method. |
| checkCreditRating | A user submits a form which contains some of his/her details, when the checkCreditRating method of the CompareLoanWS web service is invoked, another web service (CreditRating) system method will be invoked using **Service Control**. After the reply from the external web service, the result is shown to the user. |

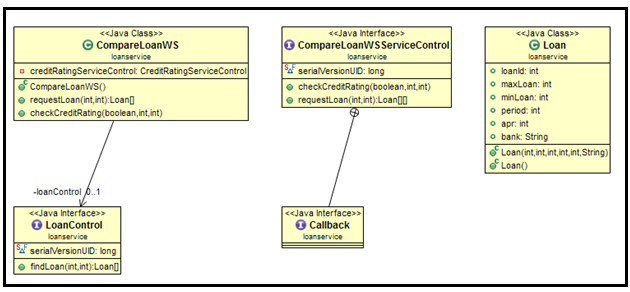


Figure 5.4 Class diagram of the CompareLoanWS System.

**Classes:**

* CompareLoanWS

This is the main class of the whole application, it contains two methods the first method is the requestLoan which takes two integers (Amount of the loan and the period of the loan) and returns an array of object type Loans. The second method is the checkCreditRating, which takes three inputs (Employment status, Salary and Insurance Number) and returns the credit rating of the particular applicant.

* Loan

This class is used to hold the data returned from querying the Loandb database in an object of its type. All member variables of this class match the table loandetails fields.

**Interfaces:**

* CompareLoanWSServiceControl

This is a service Control created from the CompareLoanWS WSDL file, that will be used to provides an interface to the client’s user interface which uses this Web Service.

* LoanControl

This is a Database Control that will provides the CompareLoanWS Web Service with the tools necessary to connect and query the Loandb database, by using its Control method findLoan.

* callback

Allows the CompareLoanWS Web Service to listen for and receive callbacks from the client API that requests a service.

## 5.5 CreditRating

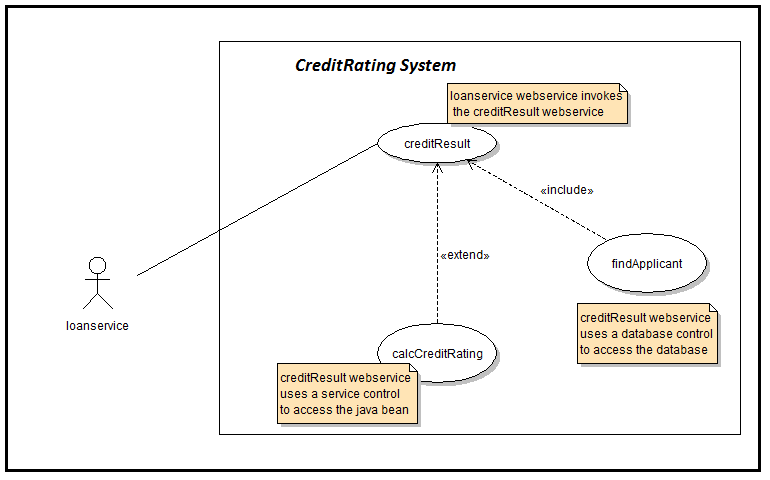
****

Figure 5.5 Use Case diagram of the CreditRating System.

The CreditRating system is an external Web Service, which is invoked by the CompareLoanWS using a Service Control. This Web Service has an in-house Bankruptcy database and EJB to calculate the credit rating of an applicant.

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| creditResult | When the CompareLoanWS invokes the creditResult method of the CreditRating Web Service Using a **Service Control**, the creditResult method invokes the findApplicant method. |
| findApplicant | The findApplicant method, which is a **Database Control method**, checks the Bankruptcy database against the Applicant NI. Then it returns true or false. |
| calcCreditRating | If the findApplicant **Database Control method** returns false, then the calcCreditRating **EJB** will calculate the applicant’s credit rating, and will return the result to the creditResult method. |

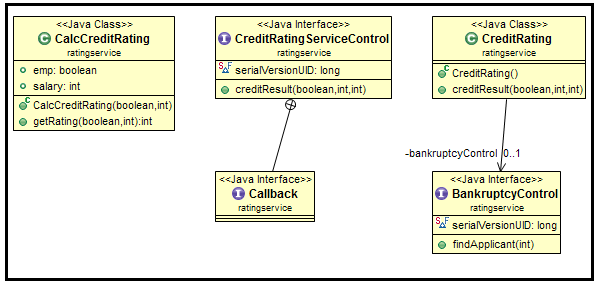
****

Figure 5.6 Class diagram of the CreditRating System.

**Classes:**

* CreditRating

Is the CreditRating Web Service main class; there is one method creditResult which returns the credit rating of an applicant. This class uses the Bankruptcy Control to query the bankruptcydb database.

* CalcCreditRating

This is EJB (Enterprise Java Beans) which is used to calculate the credit rating of an applicant based on his/her employment and salary variables, using the getRating method.

**Interfaces:**

* CreditRatingServiceControl

This is a service Control created from the CreditRating WSDL file, that will be used to provide an interface to the CompareLoanWS Web Service.

* BankruptcyControl

This is a Database Control that will provide the CreditRating Web Service with the tools necessary to connect and query the bankruptcydb database, by using its Control method findApplicant.

* callback

Allow the CreditRating Web Service to listen for and receive callbacks from the CompareLoanWS Web Service.

Figure 5.7 sequence diagram shows in details the flow and the logic of the application.

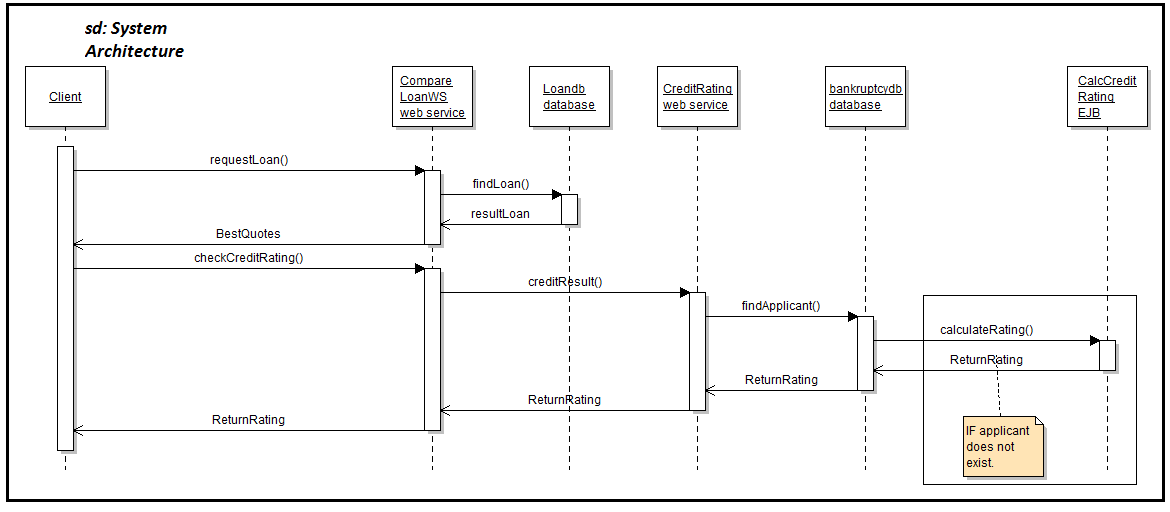
****

Figure 5.7 sequence diagram shows in details the flow and the logic of the application.

Figure 5.7 shows clearly that the Loan Comparison Application is made up of several systems, which are integrated together. This is called WSC (Web Services Composition), this paradigm is the result of an idea where a functionality of any system can be achieved, by a combination of process and systems that corporate to deliver a service.

WSC has the potential to revolutionise many industries, like E-commerce, System Integration, etc. by building more complex high level services to promote rapid application development and cross-organisational collaboration.

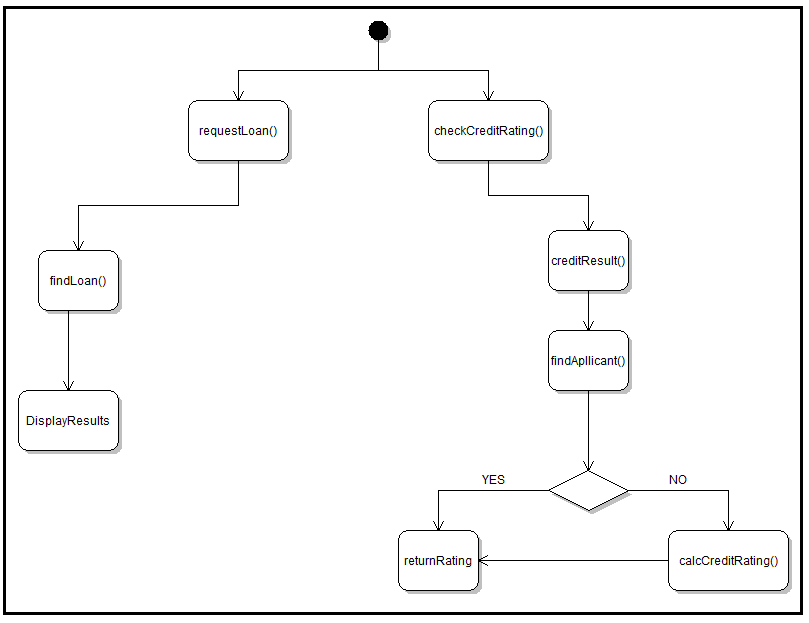
****

Figure 5.8 Activity diagram for the Loan Comparison Application.

## 5.6 Databases

The Loan Comparison Application has two databases, bankruptcydb and the Loandb databases which are part of the CreditRating system and the CompareLoanWS system respectively.

* Bankruptcydb

It’s a simple database that contains only one table bankruptcytb that holds information about applicants whom are bankrupt.

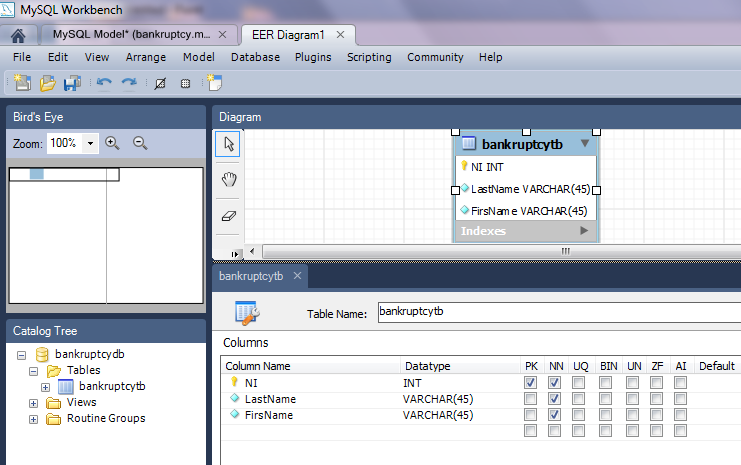


Figure 5.9 the bankruptcydb database and the bankruptcytb table.

* Loandb

It’s a simple database that contains only one table loandetails, which holds information about loans provided by major Landers.

In a real life scenario this database will be hosted on the loan providers (Banks) system, and the Loan comparison application will access these databases after establishing an agreement with the provider. Also this database in real world scenario will be updated by the loan provider.

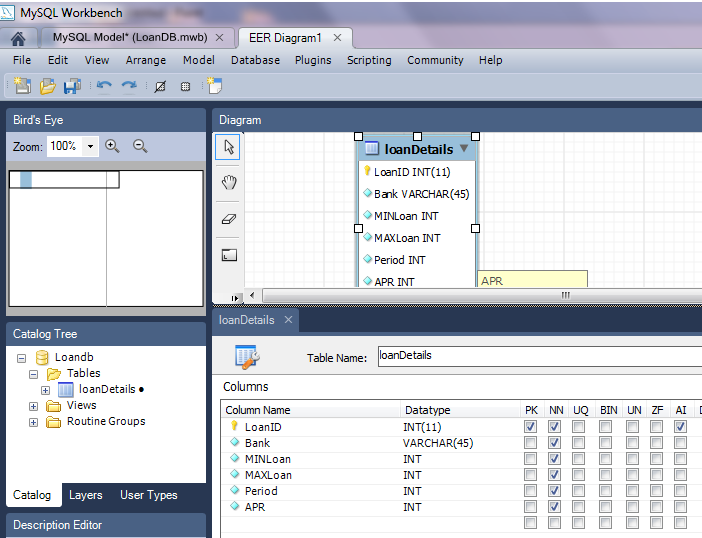


Figure 5.10 the Loandb database and the loandetails table.

## 5.7 User Interface

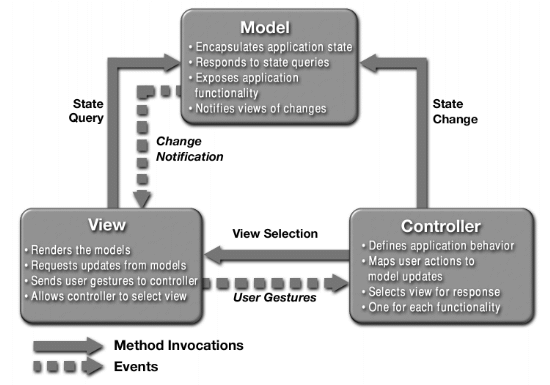
Any client application capable of communicating using XML can use the Loan Comparison Application. The whole point of building this application with Web Services technology is to be able to hide and separate the Business logic of the Loan Comparison from the client system. In order to achieve this aim, an MVC (Model-View -Controller) design pattern is applied to the user interface design.

**MVC**:

Model: represent the data layer of the application, and the logic used to manipulate this data.

View: represent the presentation layer in an application, it managed the user interface such as Web Browsers, JSPs.

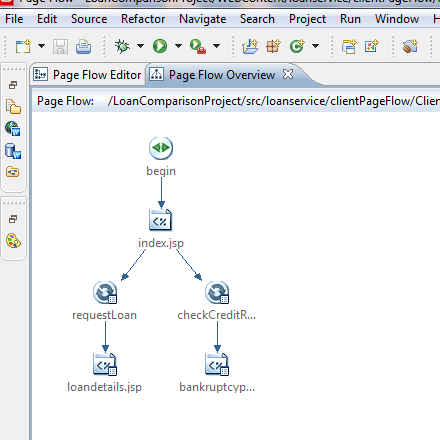
Controller: is the middle layer which manages the interaction between the user view and layer and the model layer.



[http://java.sun.com/blueprints/patterns/MVC-detailed.html](%20http:/java.sun.com/blueprints/patterns/MVC-detailed.html)

For the **Loan Comparison application** the MVC pattern is implemented by using a pageFlow technology in WebLogic environment. PageFlow, or as it is known ‘Control Framework’ is an extension of the Struts framework.

* The View in the PageFlow design of the application is represented by three JSPs (Java Server Pages). The first is the **index** page, which will be loaded when the user first accesses the **Loan Comparison Application**. The other two pages are **loandetails** and **bankruptcypage**, which represent the output of the **requestLoan** and **checkCreditRating** Web Services methods respectively.
* The Model in the application represents the Web Services **CompareLoanWS** and **CreditRating**. The representation of these Web Services is achieved by using their Service Control **CompareLoanWSServiceControl** and **CreditRatingServiceControl** respectively.
* The Controller in the user interface design is the page flow control file **ClientPageFlowController**, which has two functions: manage the navigation between the JSPs and accessing the Web Service.



## 5.6 Summary

The chapter started by analysing the application and identifying its major components (Web Services and user interface). This led to the design of each component separately, with the help of use case and class diagrams.

Also the MVC design pattern was introduced in the user interface design phase.

# Chapter 6 - Implementation

After the design stage has been completed, the implementation of the application will be introduced in this chapter. First this chapter starts by introducing all the technologies used to develop the application, as in the design stage each part or unit of the system is implemented separately before the integration.

## 6.1 Software and Tools Used

Based on the Design and Analysis stage the tools chosen to implement the artefacts are:

### 6.1.1 J2EE

Java 2 Platform Enterprise Edition is a very popular framework based on Java from Sun for developing web enterprise applications. It coveres all of Java Standard technology and more, like Servlet, JavaMail, JSF (Java Server Faces), JMS (Java Messaging Service), EJB (Enterprise Java Beans) and others.

Multi-tier system architectures can be developed by using the J2EE technology, which help business to separate their business logic and data frame from client side application. J2EE supports the MVC (Model, View and Control) approach.

### 6.1.2 WebLogic Server 10gR

WebLogic server is powerful Java Enterprise Edition application web server from Oracle, first developed and owned by BEA, and then was bought by Oracle. WebLogic server provides a standard set of APIs for developing and deploying distributed Java applications.

WebLogic server 10gR supports many Web Service standards (WSDL, SOAP, WS-Security and UDDI, etc….). In addition to all its benefits WebLogic Server supports JAX-WS (Java API for XML based Web Services) tools and WS\* implementation.

### 6.1.3 Workshop for WebLogic

The workshop for WebLogic is an IDE based on Eclipse and contains many sets of plug-ins. Workshop can be download from the Oracle website or it can come with the WebLogic server 10gR download package.

“WebLogic Workshop is the industry's leading visual development environment for building enterprise-class web services. With WebLogic Workshop, you can focus on building business logic into your web service rather than on complex implementation details. Whether you are an application developer with a problem to solve and no J2EE experience or a J2EE code jockey, WebLogic Workshop makes it easy to design, test, and deploy web services. WebLogic Workshop's intuitive user interface lets you design your web service visually. Controls make it simple to connect to enterprise resources, like databases and Enterprise JavaBeans, without writing a lot of code. Conversations handle the job of keeping track of application state. And WebLogic Workshop's support for asynchronous processes makes it easy to build highly reliable web services for the enterprise.” [11]

Workshop advantages:

* Design View: application or Web Service can be developed initially in design view, also resources connections such databases, Web Services and EJBs can be viewed in design view.
* Controls as mentioned before of all type (Database, Service, EJBs, Time and JMS) control can be developed to simplify applications and resources integration.
* XML parsing is done automatically, converting messages from Java to XML and vice versa is handled by the Workshop.

### 6.1.4 MySQL

MySQL is an open source DBMS (Database Management System), the most popular choice of database for use in web applications and E-commerce web sites. With a powerful query engine MySQL databases are scalable, fast and reliable data management systems.

## 6.2 Implementing and Connecting Databases

* Building the databases (Loandb and bankruptcydb) needed for the Loan Comparison application, as shown before in figures (5.9 and 5.10).
* Connecting the database to the application Web Services in WebLogic server, the administration tool has been used to create a connection pool. First a data source and connection need to be created.

Figure (6.1 and 6.2) shows

* The data sources names (JDBC Data Source-Bankruptcy and JDBC Data Source-Loan).
* The connections name(JNDIBankruptcy and JNDI-Loan)

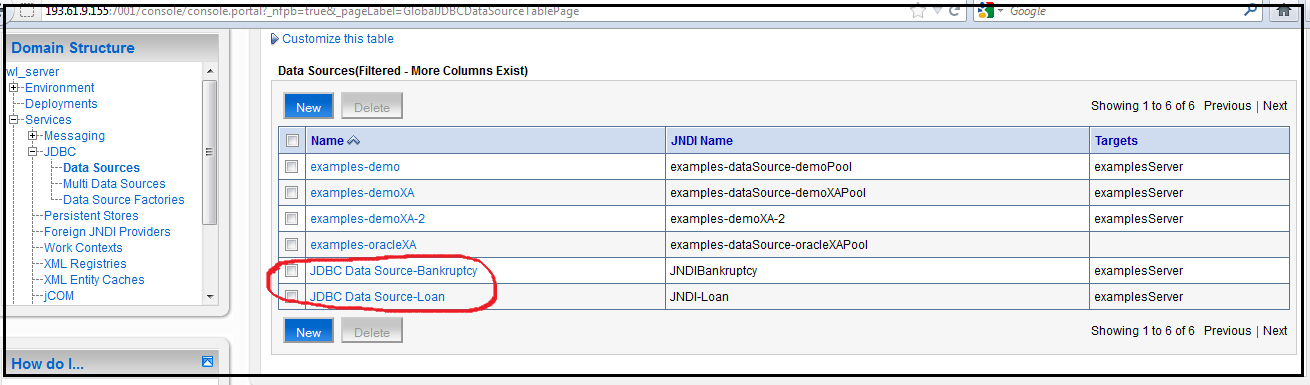


Figure 6.1 Shows the Data Source and the Connection name of the databases in WebLogic Administration tool.

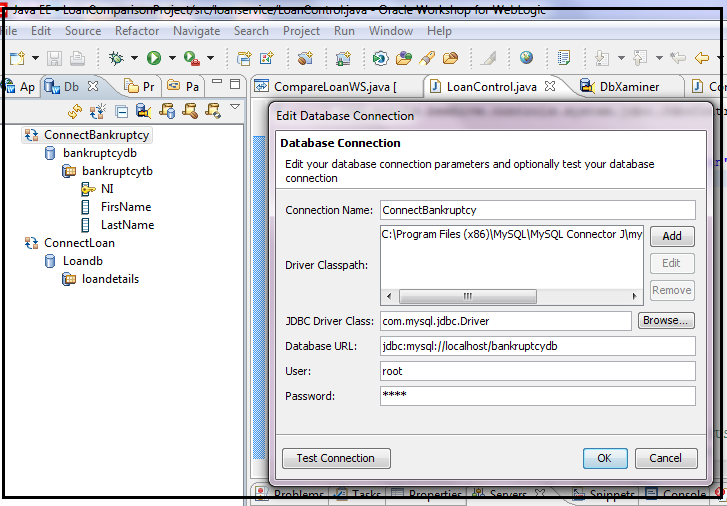


Figure 6.2 shows the Database Connection.

## 6.3 CompareLoanWS Web Service

### 6.3.1 CompareLoanWS Class

The **CompareLoanWS** Web Service is a simple Java class file. Figures 6.3 6.4 show this class and its content:

* The **CompareLoanWS** class is preceded by a **@WebService** annotation in line 5,which means that the class is a web Service class.
* There are two Controls, a Database Control **LoanControl** and a Service Control **CreditRatingServiceControl** line 10 and 16.Both of these Controls are preceded by **@Control** annotation.

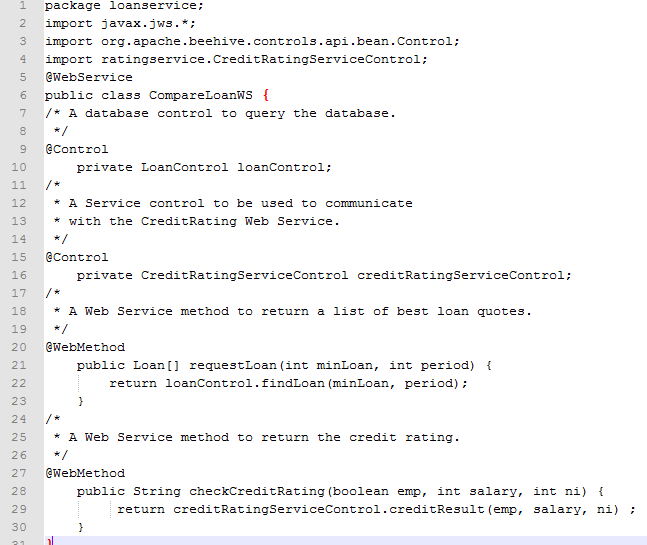


Figure 6.3 shows the CompareLoanWS class file.

* There are two methods in this class:
* **requestLoan ()** method that takes two Integers and returns an Array of Objects of type Loan.
* **checkCreditRating()** method that takes three inputs a Boolean value and two Integers, and returns a String.

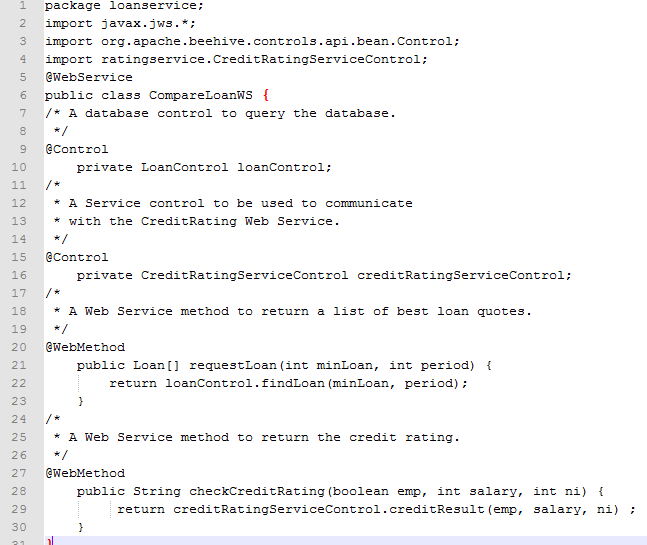


Figure 6.4 shows the CompareLoanWS class file

Both these methods are preceded by a **WebMethod** annotation.

In figure 6.5 the **CompareLoanWS** Web Service is displayed in the Workshop IDE design view.

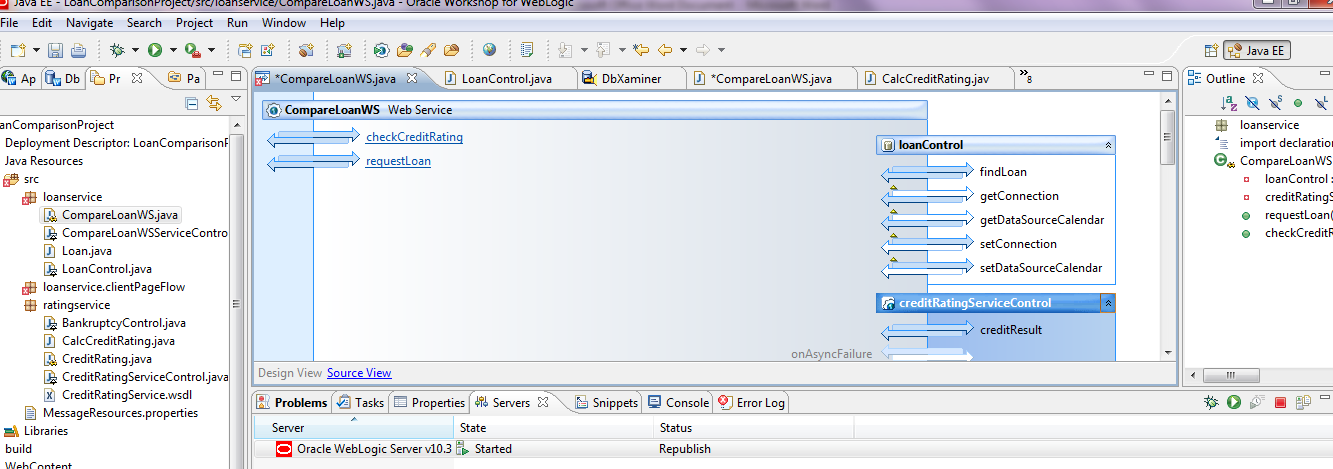


Figure 6.5 shows the CompareLoanWS Web Service in design view.

From **CompareLoanWS** class the WSDL file for the Web Service can be easily generated in WebLogic Workshop. Figure 6.6 shows small part of the WSDL file of the **CompareLoanWS** Web Service which has been generated.



Figure 6.6 shows the CompareLoanWS Web Service WSDL file.

### 6.3.2 Loan Class

The **Loan** class is a very simple Java class, which is used to create the Object that holds the loan details when querying the **Loandb** Database. The member variables of this class match the field of the **Loandb** database table **loandetails**.

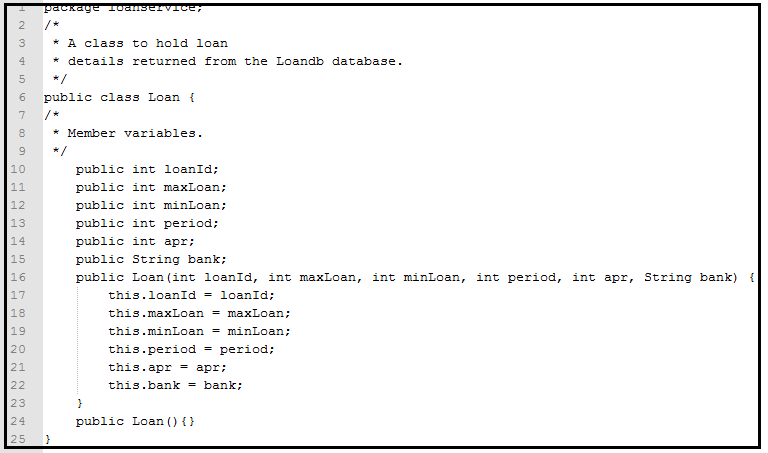


Figure 6.7 shows the class Loan.

### 6.3.3 LoanControl Interface

The Database Control **LoanControl** provides the **CompareLoanWS** Web Service with the facility to query the **Loandb** database, by calling the Java method **findLoan().** This Control automatically translates the database query in to Java object and vice-versa. Figure 6.8 6.9 shows the code for the **LoanControl**.

* Database connection is established in line 8.
* The **LoanControl** extends the **JdbcControl** class in line 9.

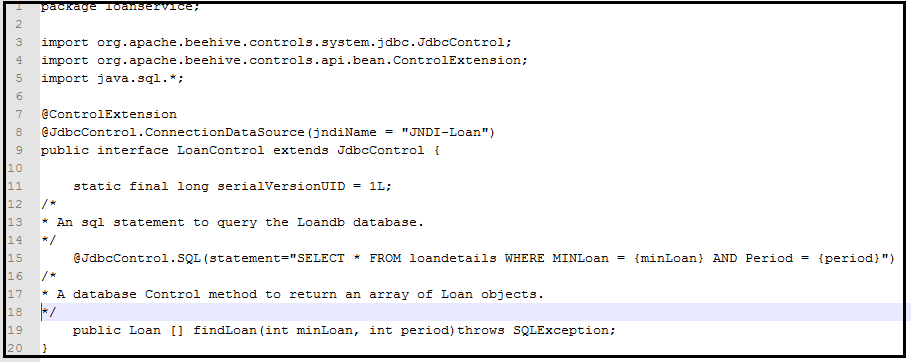


Figure 6.8 shows the Loandb database interface control LoanControl.

* Line 15 shows the SQL statement that queries the **loandetails** table.
* Finally in line 19, the **findLoan()** method, when it’s called it returns an Array of objects type **Loan**.

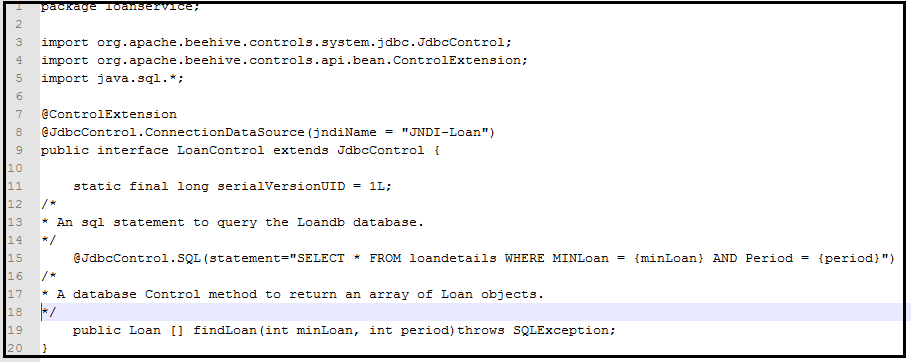


Figure 6.9 shows the Loandb database interface control LoanControl.

### 6.3.4 CompareLoanWSServiceControl Interface

The Service Control **CompareLoanWSServiceControl** provides an interface for the client application to invoke methods belonging to this Web Service.

Other Web Services that want to use the **CompareLoanWS** Web Service in future, can use this Service Control to do so, it does not matter if those Web Services are build using WebLogic Server or any other Web Server. The code in Figure 6.10 shows:

* In line 7 the location of the Web Service provided to Clients.
* The two methods that the **CompareLoanWS** Web Service offers to clients are shown in lines 16 and 18.
* Line 23 shows the **Callback** interface, which is used to handle to Web Service call-back.

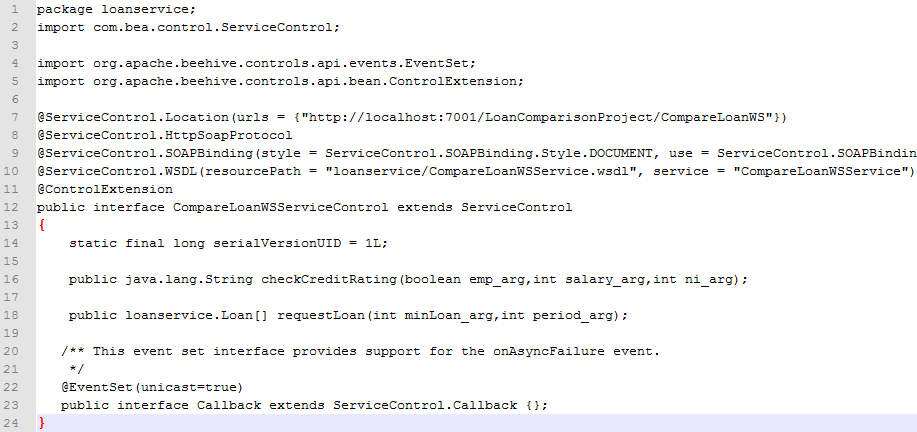


Figure 6.10 shows the CompareLoanWS Web Service Control CompareLoanWSServiceControl.

## 6.4 CreditRating Web Service

### 6.4.1 CreditRating Class

The **CreditRating** Web Service is a simple Java class file. Figure 6.9 shows this class and its content:

* The **CreditRating** class is preceded by a **@WebService** annotation in line 4,which means that the class is a web Service class.
* There is one Control, one Database Control **BankruptcyControl** inline 10, preceded by an **@Control** annotation.
* There is one methods in this class, **creditResult ()** method that takes two Integers and a Boolean value. This method is preceded by a **WebMethod** annotation.
* First this method checks if the applicant is in the **bankruptcydb** database, by using his/her NI (National Insurance).
* If the applicant is found, it returns the String variable **CreditRating** to the **CompareLoanWS** Web Service with the value bad line 19.
* Otherwise if the applicant is not found in the database, this method then calls on another method of the EJB **CalcCreditRating**. To calculate the credit rating based on the variable **salary** line 22.

In figure 6.10 the **CreditRating** Web Service is displayed in the Workshop IDE design view,



Figure 6.11 shows the CreditRating Web Service main class.

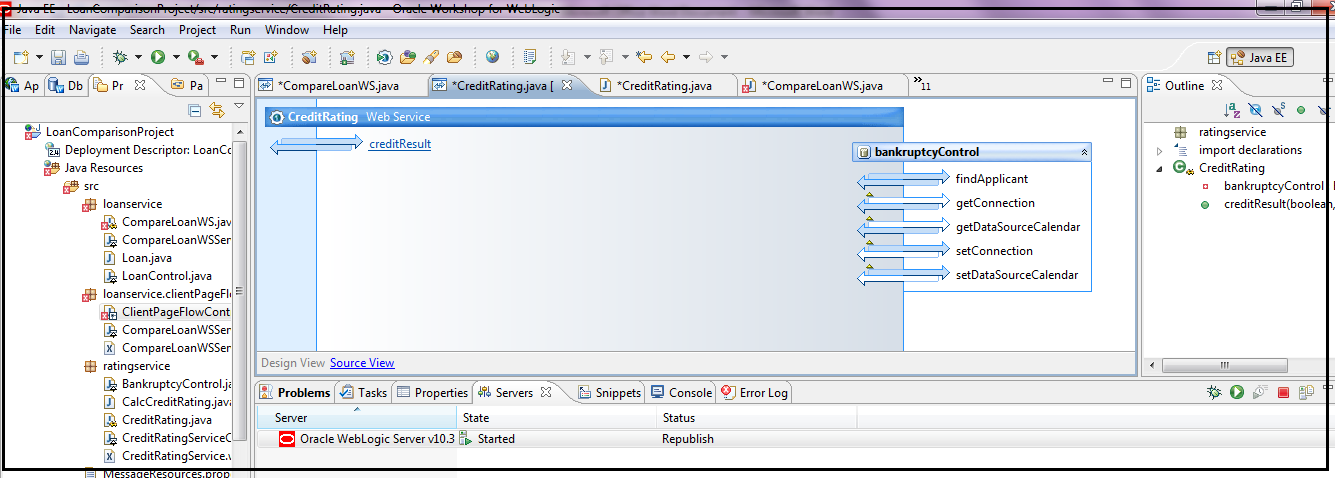


Figure 6.12 shows the CreditRating Web Service in design view.

From **CreditRating** class the WSDL file for the Web Service can be easily generated in WebLogic Workshop. Figure 6.5 shows small part of the WSDL file of the **CreditRating** Web Service which has been generated



Figure 6.13 shows the CreditRating Web Service WSDL file.

### 6.4.2 CalcCreditRating Class

The **CalcCreditRating** class is EJB Java class, which is used to calculate the credit rating of an applicant. It contain two member variables **emp** and **salary**, and a method **getRating()** that’s runs a set of if statements to determine the credit rating based on the values of the **emp** and the **salary** variables.

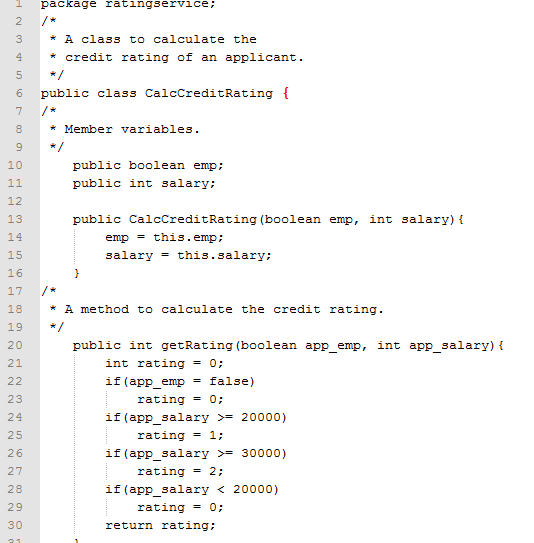


Figure 6.14 shows the EJB class CalcCreditRating.

### 6.4.3 BankruptcyControl Interface

The Database Control **BankruptcyControl** provides the **CreditRating** Web Service with the facility to query the **bankruptcydb** database, by calling the Java method **findApplicant()**.This Control automatically translates the database query in to Java objects and vice-versa. Figure 6.13 shows the code for the **BankruptcyControl**.

* Database connection is established in line 6.
* The **LoanControl** extends the **JdbcControl** class in line 7.
* Line 13 shows the SQL statement that queries the **loandetails** table.
* Finally in line 19, the **findApplicant()** method, when it’s called it returns a String which has the value of the applicant Last Name.

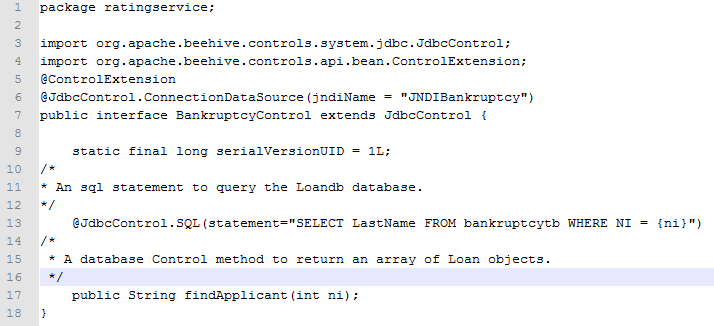


Figure 6.15 shows the bankruptcydbl database interface control BankruptcyContol.

### 6.4.4 CreditRtingServiceControl Interface

The Service Control **CreditRatingServiceControl** provides an interface for the CompareLoanWS Web Service, to invoke methods belonging to this Web Service.

Other Web Services that want to use the **CompareLoanWS** Web Service in future, can use this Service Control to do so, it does not matter if those Web Services are build using WebLogic Server or any Web Server. The code in Figure 6.14 shows:

* In line 7 the location of the Web Service is provided to Clients.
* The method that the **CreditRating** Web Service offers to clients (including the **CompareLoanWS** Web Service) is shown in line 16.
* Line 21 shows the **Callback** interface, which is used to handle to Web Service call-back.

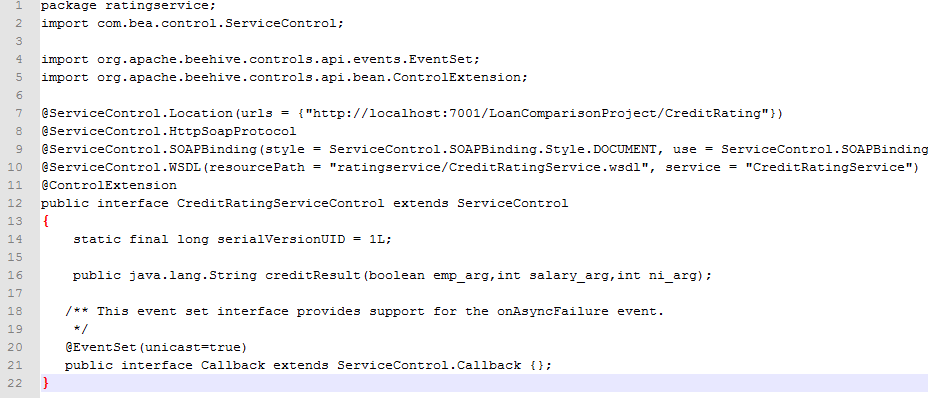


Figure 6.16 shows the CreditRating Web Service Control CreditRatingServiceControl.

## 6.5 User Interface

As mentioned in the design stage, the user interface is developed and represented by PageFlow to insure the MVC design pattern is implemented.

### 6.5.1 ClientPageFlowController

The ClientPageFlowController is a java class, which consist of few Methods, Actions and Forms.

* The **ClientPageFlowController** class extend the **PageFlowController** class, line11. Figure 6.17
* There are Three Actions:
* The first Action is the **begin** action in line 10, which is used to load the index JSP (Java Server Page). Figure 6.17



Figure 6.17 shows the ClientPageFowCotroller class.

* The second one is the Action Method **requestLoan** in line17**,** which invoke and retrieves the **CompareLoanWS** Web Service data request.



Figure 6.18 shows the ClientPageFowCotroller class requestLoan action method.

* The third Action is **CheckCreditRating** in line 32; it’s used to retrieves data from the **CreditRating** Web Service.

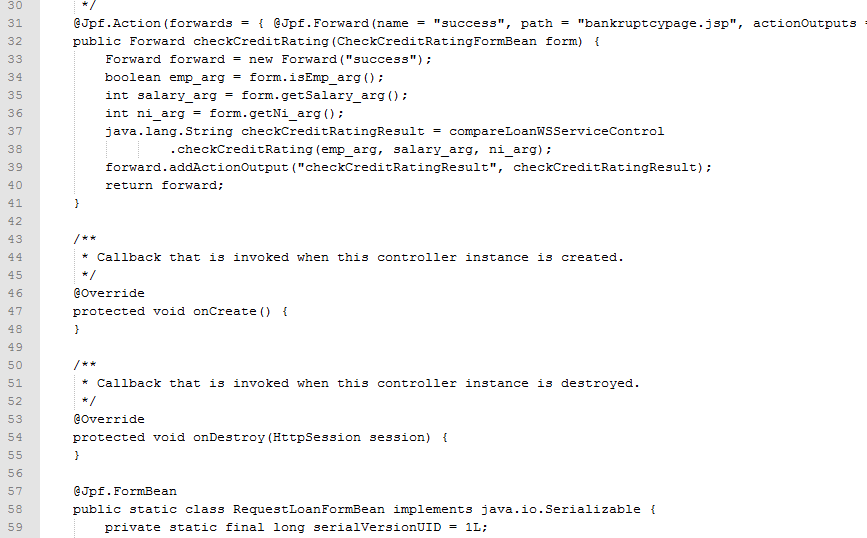


Figure 6.19 shows the ClientPageFowCotroller class checkCreditRating action method.

* The PageFlow class uses the **CompareLoanWSServiceControl** of the **CompareLoanWS** Web Service. In line14.



Figure 6.20 shows the ClientPageFowCotroller class Control.

* The **create()** and **onDestroy()** Methods are both a **Servlet HttpSession** methods, which are used to control the instant of the session when a client/user interact with the **Loan Comparison Application**.
* The **ClientPageFlowController** class has two Form Beans:
* **requestLoanFormBean** is used to collect user input to pass it to the **CompareLoanWS** Web Service.
* **checkCreditRatingFormBean**  is used to collect user inputs and pass it to **CreditRating** Web Service.

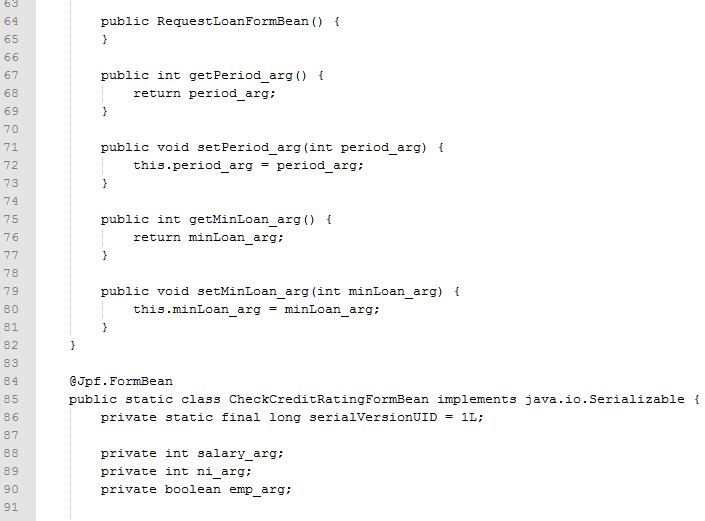


Figure 6.21 shows the ClientPageFowCotroller class FormBeans.

### 6.5.2 JSPs (Java Server Pages)

The index page is a basic HTML page, which contains some basic HTML tags and two forms.

The first form, requestLoan, corresponds to the ClientPageFlowController form requestLoanFormBean. The second form checkCreditRating corresponds to the checkCreditRatingFormBean.

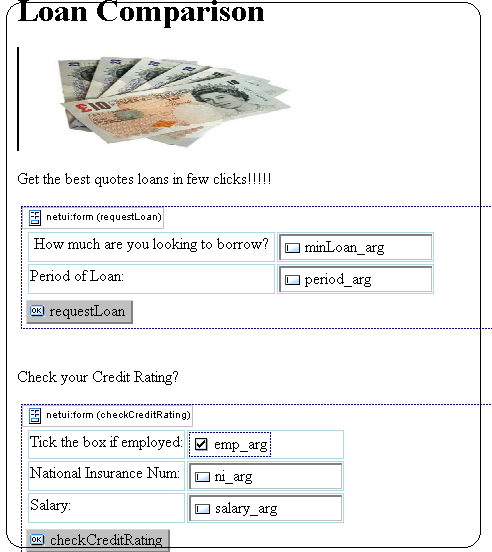


Figure 6.18 shows the index page in design view.

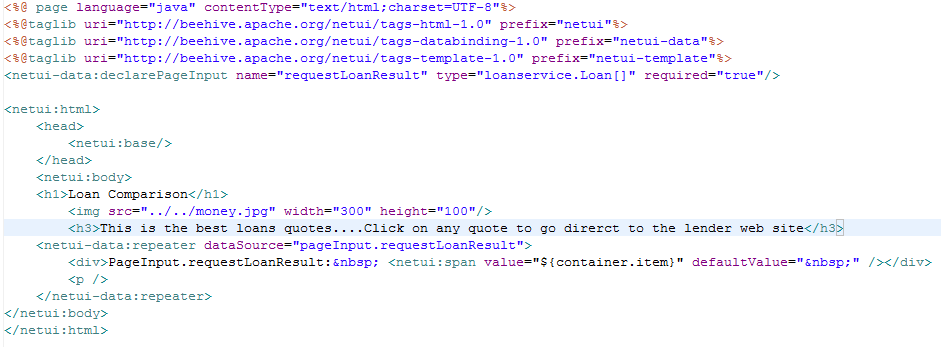


Figure 6.19 shows the loandetails JSP page in source view.

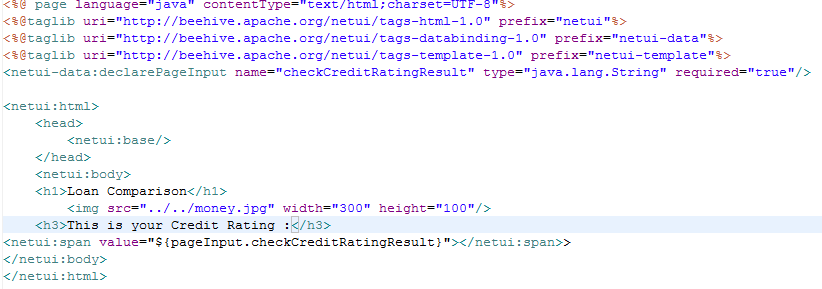


Figure 6.20 shows the bankruptcy JSP page in source view.

## 6.6 – Summary

In this chapter the LoanComparison Application was implemented. This application system architecture was based on Web Services, therefore each separate component was treated as a separate unit.

First the CompareLoanWS and CreditRating Web Services were developed and implemented separately. Following this stage the two Web Services were integrated together, finally the user interface was developed and integrated in to the System.

# Chapter 7- Testing

Based on Incremental development methodology chosen to develop the Loan Comparison Application, the testing phase was performed at different stages of the application development lifecycle. Also the nature of Web Services technology that is based on the idea of Loose Coupling, each component of the System can be regarded as separate system or unit in its own.

Therefore the first testing will be carried out using **Black Box** testing, to test each system just after completing its development.

WebLogic server development environments come with build in **Test Client Unit**, which will be used in each testing stage.

## 7.1 – CreditRating Web Service Testing

The CreditRating web service consists of a Web Service class **CreditRating,** a Database Control **bankruptcyControl** and an EJB class **CalcCreditRating**. This system returns an applicant credit rating based on some variables (applicant salary and applicant employment status).

Figure 7.1 shows the output of invoking the Web Method **creditResult()** of the **CreditRating** Web Service, when it’s invoked with the parameters values (emp = true and salary = 20000).

Figure 7.1 shows the output of invoking the creditResult() Web Method.

## 7.2 – CompareLoanWS Web Service Testing

The **CompareLoanWS** Web Service class **CompareLoanWS** consists of two Web Methods **requestLoan()** and **checkCreditRating()**.

Figure 7.2 shows the output of invoking the Web Method **requestLoan()** with parameters values (minLoan = 1000 and period = 6).

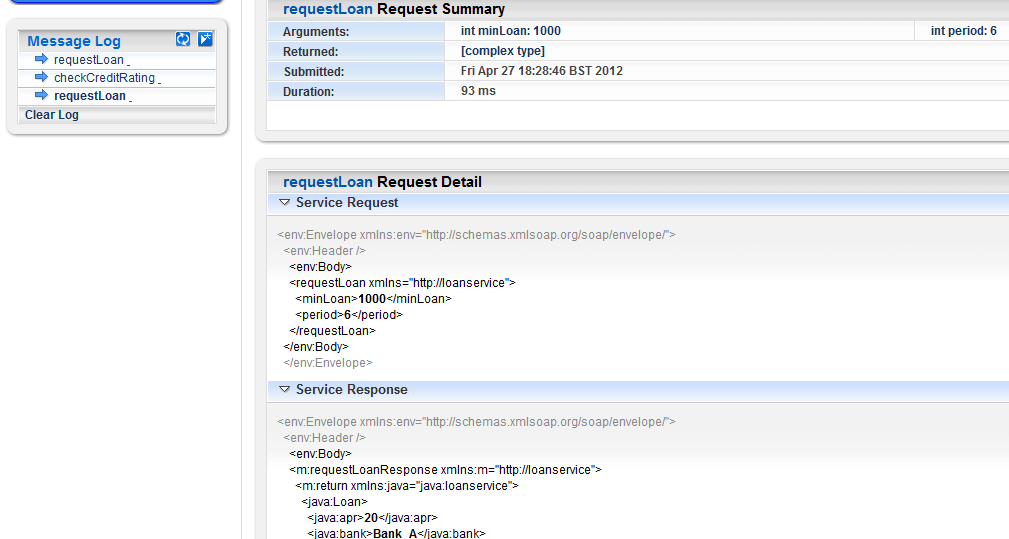




Figure 7.2 shows the output of invoking the requestLoan() Web Method.

Figure 7.3 shows the output of invoking the Web Method **checkCreditRating()** with parameters values (emp = true and salary = 20000).

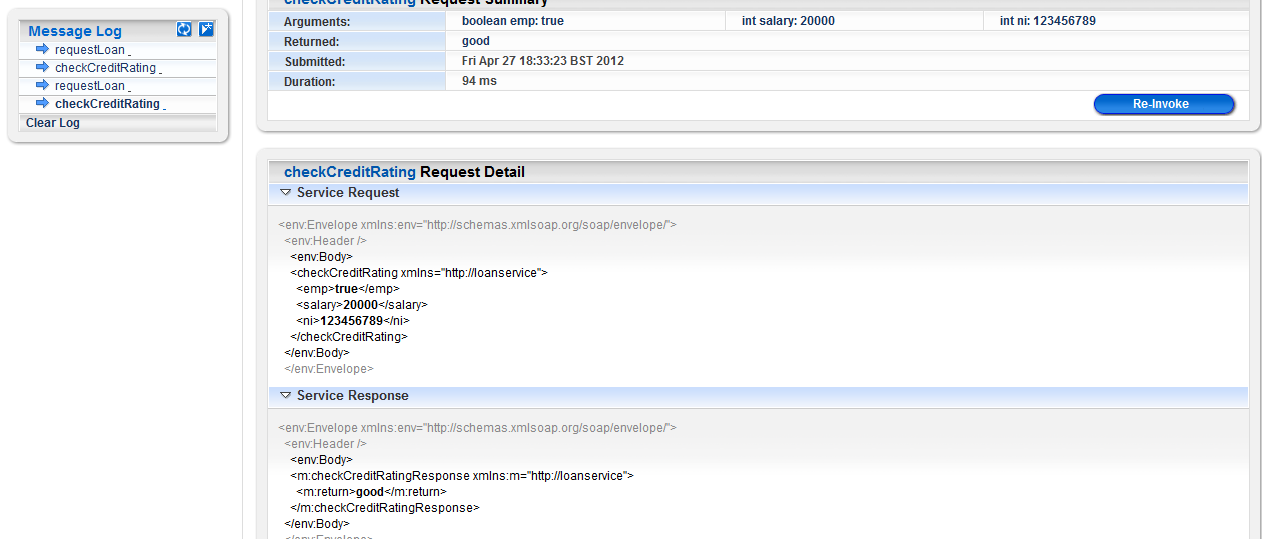


Figure 7.1 shows the output of invoking the checkCreditRating() Web Method.

## 7.3 – User Testing

Upon completion of the development of the Loan Comparison Application, 3 users have been asked to test and use this application and to comment on their experience with feedback.

|  |  |
| --- | --- |
| **User task** | **Comment** |
| Loading the application | The application loaded in reasonable time, but the URL used is too complex to remember (http:/localhost:7001/LoanComparisonProject/CompareLoanWS?WSDL). |
| If there is help option | There are no help options or read me menu how to use the application. |
| Simple and easy interface | Even the Interface is simple to use, the application should have an attractive GUI to go with it. |
| Instant output | The output is instant, but lacks explanation of meaning of the output. |
| security | The application uses HTTP web protocol, there is no security implementation in place. |
| General experience by visiting the web site | Even though the system does what it supposed to do, the application is very simple and can be developed in better way.. |

# Chapter 8 - Conclusion

The aim of this project was to develop a Loan Comparison Application to explore the Web Services technologies and its architecture. A considerable amount of time was spent in research and trying to understand some of tools and platforms used to develop such an application.

Especially great amounts of time were designated for researching the WebLogic server environment technology, including Control (Database Control, Service Control and EJB Control), and PageFlow Struts framework. The Oracle WebLogic development environment is a very useful tool to develop Web Services based applications; it contains many features and tools. More time was needed to take advantage of what WebLogic offers.

The application development was based on the Object Oriented (OO) approach, therefore the application components are loosely coupled, reusable and interoperable. New systems can be added or integrated with the Loan Comparison Application.

Until the closing stages of the development of the Loan Comparison Application, the full benefit of Web Services technology and the advantages that it could bring in E-commerce is clear. Web Services are the bed rock of many technologies and approaches in distributed systems. The rise of SOA (Service Oriented Approach) and cloud computing in these days would not be happening if it wasn’t for Web Services.

Due to time constraints the user interface for the system was not developed completely using PageFlow. Another aspect missing from the development is the security, even though the application developed in this project was supposed to offer information to users only.

# REFERENCES

[1] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris

Ferris and David Orchard, "Web Services Architecture," Available:

http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. [Accessed: 23 February 2007].

*[2] - [http://www.w3.org/2002/ws/ - World Wide Web Consortium - Web Services](http://www.w3.org/2002/ws/%20-%20World%20Wide%20Web%20Consortium%20-%20Web%20Services)*

*[3] -* [*http://docs.oracle.com/cd/E13226\_01/workshop/docs70/help/index.html#guide/tutorial/tutFirstWebServiceStep1.html*](http://docs.oracle.com/cd/E13226_01/workshop/docs70/help/index.html%23guide/tutorial/tutFirstWebServiceStep1.html)

*[4]-* [*http://uddi.xml.org/*](http://uddi.xml.org/)

*[5]-* [*http://www.w3schools.com/wsdl/wsdl\_uddi.asp*](http://www.w3schools.com/wsdl/wsdl_uddi.asp)

*[6] -* [*http://www.altova.com/whitepapers/webservices.pdf*](http://www.altova.com/whitepapers/webservices.pdf)

*[7] – Simon Bennet, Steve McRobb and RayFarmer, “Object-Oriented System Analysis And Design Using UML”. Third Edition Nc Graw Hill. Chapter 21, Page 604.*

*[8] – Simon Bennet, Steve McRobb and RayFarmer, “Object-Oriented System Analysis And Design Using UML”. Third Edition Nc Graw Hill. Chapter 3, Page 56.*

[9]- h[ttp://msdn.microsoft.com/en-us/library/aa302336.aspx](file:///C:\\Users\\Mahfoud\\Downloads\\Mozilla%20Firefox.lnk)

[10]- <http://www.edm2.com/0601/rmi1.html>

[11][http://docs.oracle.com/cd/E13226\_01/workshop/docs70/help/index.html#guide/tutorial/tutFirstWebServiceStep1.html](http://docs.oracle.com/cd/E13226_01/workshop/docs70/help/index.html%23guide/tutorial/tutFirstWebServiceStep1.html)

# Appendix

## 1 - CompareLoanWS Web Service

### 1.1 CompareLoanWS class

package loanservice;

import javax.jws.\*;

import org.apache.beehive.controls.api.bean.Control;

import ratingservice.CreditRatingServiceControl;

@WebService

public class ComparisonLoanWS {

@Control

private LoanControl loanControl;

@Control

private CreditRatingServiceControl creditRatingServiceControl;

@WebMethod

public Loan[] requestLoan(int minLoan, int period) {

return loanControl.findLoan(minLoan, period);

}

@WebMethod

public String checkCreditRating(boolean emp, int salary, String ni) {

return creditRatingServiceControl.creditResult(emp, salary, ni);

}

}

1.2 Loan class

**package** loanservice;

**public** **class** Loan {

**public** **int** loanId;

**public** **int** maxLoan;

**public** **int** minLoan;

**public** **int** period;

**public** **int** apr;

**public** String bank;

/\*\*

\* **@param** loanId

\* **@param** maxLoan

\* **@param** minLoan

\* **@param** period

\* **@param** apr

\* **@param** bank

\*/

**public** **int** getLoanId() {

**return** loanId;

}

**public** **void** setLoanId(**int** loanId) {

**this**.loanId = loanId;

}

**public** **int** getMaxLoan() {

**return** maxLoan;

}

**public** **void** setMaxLoan(**int** maxLoan) {

**this**.maxLoan = maxLoan;

}

**public** **int** getMinLoan() {

**return** minLoan;

}

**public** **void** setMinLoan(**int** minLoan) {

**this**.minLoan = minLoan;

}

**public** **int** getPeriod() {

**return** period;

}

**public** **void** setPeriod(**int** period) {

**this**.period = period;

}

**public** **int** getApr() {

**return** apr;

}

**public** **void** setApr(**int** apr) {

**this**.apr = apr;

}

**public** String getBank() {

**return** bank;

}

**public** **void** setBank(String bank) {

**this**.bank = bank;

}

}

### 1.3 ComparisonLoanWSServiceContro interface

package loanservice;

import com.bea.control.ServiceControl;

import org.apache.beehive.controls.api.events.EventSet;

import org.apache.beehive.controls.api.bean.ControlExtension;

@ServiceControl.Location(urls = {"http://localhost:7001/LoanComparisomProject/ComparisonLoanWS"})

@ServiceControl.HttpSoapProtocol

@ServiceControl.SOAPBinding(style = ServiceControl.SOAPBinding.Style.DOCUMENT, use = ServiceControl.SOAPBinding.Use.LITERAL, parameterStyle = ServiceControl.SOAPBinding.ParameterStyle.WRAPPED)

@ServiceControl.WSDL(resourcePath = "loanservice/ComparisonLoanWSService.wsdl", service = "ComparisonLoanWSService")

@ControlExtension

public interface ComparisonLoanWSServiceControl extends ServiceControl

{

static final long serialVersionUID = 1L;

public java.lang.String checkCreditRating(boolean emp\_arg,int salary\_arg,java.lang.String ni\_arg);

public loanservice.Loan[] requestLoan(int minLoan\_arg,int period\_arg);

/\*\* This event set interface provides support for the onAsyncFailure event.

\*/

@EventSet(unicast=true)

public interface Callback extends ServiceControl.Callback {};

}

### 1.4 LoanControl interface

**package** loanservice;

**import** org.apache.beehive.controls.system.jdbc.JdbcControl;

**import** org.apache.beehive.controls.api.bean.ControlExtension;

/\*\*

\* **@author** Mahfoud

\*

\*/

@ControlExtension

@JdbcControl.ConnectionDataSource(jndiName = "JNDI-Loan")

**public** **interface** LoanControl **extends** JdbcControl {

**static** **final** **long** *serialVersionUID* = 1L;

@JdbcControl.SQL(statement="SELECT \* FROM loandetails WHERE MINLoan = {minLoan} AND Period = {period}")

**public** Loan[] findLoan(**int** minLoan, **int** period);

}

### 1.5 ComparisonLoanWSService WSDL file

<?xml version='1.0' encoding='UTF-8'?>

<s0:definitions name="ComparisonLoanWSServiceDefinitions" targetNamespace="http://loanservice" xmlns="" xmlns:s0="http://schemas.xmlsoap.org/wsdl/" xmlns:s1="http://loanservice" xmlns:s2="http://schemas.xmlsoap.org/wsdl/soap/">

<s0:types>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="java:loanservice" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:complexType name="Loan">

<xs:sequence>

<xs:element minOccurs="1" name="apr" nillable="false" type="xs:int"/>

<xs:element minOccurs="1" name="bank" nillable="true" type="xs:string"/>

<xs:element minOccurs="1" name="loanId" nillable="false" type="xs:int"/>

<xs:element minOccurs="1" name="maxLoan" nillable="false" type="xs:int"/>

<xs:element minOccurs="1" name="minLoan" nillable="false" type="xs:int"/>

<xs:element minOccurs="1" name="period" nillable="false" type="xs:int"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="ArrayOfLoan\_literal">

<xs:sequence>

<xs:element maxOccurs="unbounded" minOccurs="0" name="Loan" nillable="true" type="java:Loan" xmlns:java="java:loanservice"/>

</xs:sequence>

</xs:complexType>

<xs:element name="ArrayOfLoan\_literal" type="java:ArrayOfLoan\_literal" xmlns:java="java:loanservice"/>

</xs:schema>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://loanservice" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:import namespace="java:loanservice"/>

<xs:element name="requestLoan">

<xs:complexType>

<xs:sequence>

<xs:element name="minLoan" type="xs:int"/>

<xs:element name="period" type="xs:int"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="requestLoanResponse">

<xs:complexType>

<xs:sequence>

<xs:element name="return" type="java:ArrayOfLoan\_literal" xmlns:java="java:loanservice"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="checkCreditRating">

<xs:complexType>

<xs:sequence>

<xs:element name="emp" type="xs:boolean"/>

<xs:element name="salary" type="xs:int"/>

<xs:element name="ni" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="checkCreditRatingResponse">

<xs:complexType>

<xs:sequence>

<xs:element name="return" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

</s0:types>

<s0:message name="requestLoan">

<s0:part element="s1:requestLoan" name="parameters"/>

</s0:message>

<s0:message name="requestLoanResponse">

<s0:part element="s1:requestLoanResponse" name="parameters"/>

</s0:message>

<s0:message name="checkCreditRating">

<s0:part element="s1:checkCreditRating" name="parameters"/>

</s0:message>

<s0:message name="checkCreditRatingResponse">

<s0:part element="s1:checkCreditRatingResponse" name="parameters"/>

</s0:message>

<s0:portType name="ComparisonLoanWS">

<s0:operation name="requestLoan" parameterOrder="parameters">

<s0:input message="s1:requestLoan"/>

<s0:output message="s1:requestLoanResponse"/>

</s0:operation>

<s0:operation name="checkCreditRating" parameterOrder="parameters">

<s0:input message="s1:checkCreditRating"/>

<s0:output message="s1:checkCreditRatingResponse"/>

</s0:operation>

</s0:portType>

<s0:binding name="ComparisonLoanWSServiceSoapBinding" type="s1:ComparisonLoanWS">

<s2:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<s0:operation name="requestLoan">

<s2:operation style="document"/>

<s0:input>

<s2:body parts="parameters" use="literal"/>

</s0:input>

<s0:output>

<s2:body parts="parameters" use="literal"/>

</s0:output>

</s0:operation>

<s0:operation name="checkCreditRating">

<s2:operation style="document"/>

<s0:input>

<s2:body parts="parameters" use="literal"/>

</s0:input>

<s0:output>

<s2:body parts="parameters" use="literal"/>

</s0:output>

</s0:operation>

</s0:binding>

<s0:service name="ComparisonLoanWSService">

<s0:port binding="s1:ComparisonLoanWSServiceSoapBinding" name="ComparisonLoanWSSoapPort">

<s2:address location="http://localhost:7001/LoanComparisomProject/ComparisonLoanWS"/>

</s0:port>

</s0:service>

</s0:definitions>

## 2 – CreditRating Web Service

### 2.1 CreditRating class

**package** ratingservice;

**import** javax.jws.\*;

**import** org.apache.beehive.controls.api.bean.Control;

@WebService

**public** **class** CreditRating {

@Control

**private** BankruptcyControl bankruptcyControl;

@WebMethod

**public** String creditResult(**boolean** emp, **int** salary, String ni) {

String creditRating = "";

**if**((emp ==**true** || emp ==**false**) && (salary > 0) && (ni.length() == 9)){

**if**(bankruptcyControl.findApplicant(ni) != **null**)

creditRating = "bad";

**else** {

CalcCreditRating c\_rating = **new** CalcCreditRating(emp, salary);

**int** result = c\_rating.getRating(emp, salary);

**if**(result == 0)

creditRating = "bad";

**if**(result == 1)

creditRating = "good";

**if**(result == 2)

creditRating = "very good";

}

}

**else**

creditRating = "invalid input";

**return** creditRating;

}

}

### 2.2 CreditRatingServiceControl interface

package ratingservice;

import com.bea.control.ServiceControl;

import org.apache.beehive.controls.api.events.EventSet;

import org.apache.beehive.controls.api.bean.ControlExtension;

/\*\*

\* @author Mahfoud

\*

\*/

@ServiceControl.Location(urls = {"http://localhost:7001/LoanComparisomProject/CreditRating"})

@ServiceControl.HttpSoapProtocol

@ServiceControl.SOAPBinding(style = ServiceControl.SOAPBinding.Style.DOCUMENT, use = ServiceControl.SOAPBinding.Use.LITERAL, parameterStyle = ServiceControl.SOAPBinding.ParameterStyle.WRAPPED)

@ServiceControl.WSDL(resourcePath = "ratingservice/CreditRatingService.wsdl", service = "CreditRatingService")

@ControlExtension

public interface CreditRatingServiceControl extends ServiceControl

{

static final long serialVersionUID = 1L;

public java.lang.String creditResult(boolean emp\_arg,int salary\_arg,java.lang.String ni\_arg);

/\*\* This event set interface provides support for the onAsyncFailure event.

\*/

@EventSet(unicast=true)

public interface Callback extends ServiceControl.Callback {};

}

### 2.3 CalcCreditRating class

**package** ratingservice;

/\*\*

\* **@author** Mahfoud

\*

\*/

**public** **class** CalcCreditRating {

**public** **boolean** emp;

**public** **int** salary;

**public** CalcCreditRating(**boolean** emp, **int** salary){

emp = **this**.emp;

salary = **this**.salary;

}

**public** **int** getRating(**boolean** app\_emp, **int** app\_salary){

**int** rating = 0;

**if**(app\_emp = **false**)

rating = 0;

**if**(app\_salary >= 20000)

rating = 1;

**if**(app\_salary >= 30000)

rating = 2;

**if**(app\_salary < 20000)

rating = 0;

**return** rating;

}

}

### 2.4 BankruptcyControl interface

**package** ratingservice;

**import** org.apache.beehive.controls.system.jdbc.JdbcControl;

**import** org.apache.beehive.controls.api.bean.ControlExtension;

/\*\*

\* **@author** Mahfoud

\*

\*/

@ControlExtension

@JdbcControl.ConnectionDataSource(jndiName = "JNDIBankruptcy")

**public** **interface** BankruptcyControl **extends** JdbcControl {

**static** **final** **long** *serialVersionUID* = 1L;

@JdbcControl.SQL(statement="SELECT SurName FROM bankruptcytb WHERE NI = {ni}")

**public** String findApplicant(String ni);

}

### 2.5 CreditRatingService WSDL file

<?xml version='1.0' encoding='UTF-8'?>

<s0:definitions name="CreditRatingServiceDefinitions" targetNamespace="http://ratingservice" xmlns="" xmlns:s0="http://schemas.xmlsoap.org/wsdl/" xmlns:s1="http://ratingservice" xmlns:s2="http://schemas.xmlsoap.org/wsdl/soap/">

<s0:types>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" targetNamespace="http://ratingservice" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="creditResult">

<xs:complexType>

<xs:sequence>

<xs:element name="emp" type="xs:boolean"/>

<xs:element name="salary" type="xs:int"/>

<xs:element name="ni" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="creditResultResponse">

<xs:complexType>

<xs:sequence>

<xs:element name="return" type="xs:string"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

</s0:types>

<s0:message name="creditResult">

<s0:part element="s1:creditResult" name="parameters"/>

</s0:message>

<s0:message name="creditResultResponse">

<s0:part element="s1:creditResultResponse" name="parameters"/>

</s0:message>

<s0:portType name="CreditRating">

<s0:operation name="creditResult" parameterOrder="parameters">

<s0:input message="s1:creditResult"/>

<s0:output message="s1:creditResultResponse"/>

</s0:operation>

</s0:portType>

<s0:binding name="CreditRatingServiceSoapBinding" type="s1:CreditRating">

<s2:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

<s0:operation name="creditResult">

<s2:operation style="document"/>

<s0:input>

<s2:body parts="parameters" use="literal"/>

</s0:input>

<s0:output>

<s2:body parts="parameters" use="literal"/>

</s0:output>

</s0:operation>

</s0:binding>

<s0:service name="CreditRatingService">

<s0:port binding="s1:CreditRatingServiceSoapBinding" name="CreditRatingSoapPort">

<s2:address location="http://localhost:7001/LoanComparisomProject/CreditRating"/>

</s0:port>

</s0:service>

</s0:definitions>

## 3 – User Interface

### 3.1 ClientPageFlowController class

package clientPageFlow;

import javax.servlet.http.HttpSession;

import org.apache.beehive.netui.pageflow.PageFlowController;

import org.apache.beehive.netui.pageflow.annotations.Jpf;

import org.apache.beehive.controls.api.bean.Control;

import loanservice.ComparisonLoanWSServiceControl;

import org.apache.beehive.netui.pageflow.Forward;

/\*\*

\* @author Mahfoud

\*

\*/

@Jpf.Controller(simpleActions = { @Jpf.SimpleAction(name = "begin", path = "index.jsp") })

public class ClientPageFlowController extends PageFlowController {

private static final long serialVersionUID = 1L;

@Control

private ComparisonLoanWSServiceControl comparisonLoanWSServiceControl;

@Jpf.Action(forwards = { @Jpf.Forward(name = "success", path = "creditRatingpage.jsp", actionOutputs = { @Jpf.ActionOutput(name = "checkCreditRatingResult", type = java.lang.String.class) }) })

public Forward checkCreditRating(CheckCreditRatingFormBean form) {

Forward forward = new Forward("success");

boolean emp\_arg = form.isEmp\_arg();

int salary\_arg = form.getSalary\_arg();

java.lang.String ni\_arg = form.getNi\_arg();

java.lang.String checkCreditRatingResult comparisonLoanWSServiceControl

checkCreditRating(emp\_arg, salary\_arg, ni\_arg);

forward.addActionOutput("checkCreditRatingResult", checkCreditRatingResult);

return forward;

}

@Jpf.Action(forwards = { @Jpf.Forward(name = "success", path = "loanDetailsPage.jsp", actionOutputs = { @Jpf.ActionOutput(name = "requestLoanResult", type = loanservice.Loan[].class) }) })

public Forward requestLoan(RequestLoanFormBean form) {

Forward forward = new Forward("success");

int minLoan\_arg = form.getMinLoan\_arg();

int period\_arg = form.getPeriod\_arg();

loanservice.Loan[] requestLoanResult = comparisonLoanWSServiceControl

.requestLoan(minLoan\_arg, period\_arg);

forward.addActionOutput("requestLoanResult", requestLoanResult);

return forward;

}

/\*\*

\* Callback that is invoked when this controller instance is created.

\*/

@Override

protected void onCreate() {

}

/\*\*

\* Callback that is invoked when this controller instance is destroyed.

\*/

@Override

protected void onDestroy(HttpSession session) {

}

@Jpf.FormBean

public static class CheckCreditRatingFormBean implements java.io.Serializable {

private static final long serialVersionUID = 1L;

private int salary\_arg;

private java.lang.String ni\_arg;

private boolean emp\_arg;

public CheckCreditRatingFormBean() {

}

public int getSalary\_arg() {

return salary\_arg;

}

public void setSalary\_arg(int salary\_arg) {

this.salary\_arg = salary\_arg;

}

public java.lang.String getNi\_arg() {

return ni\_arg;

}

public void setNi\_arg(java.lang.String ni\_arg) {

this.ni\_arg = ni\_arg;

}

public boolean isEmp\_arg() {

return emp\_arg;

}

public void setEmp\_arg(boolean emp\_arg) {

this.emp\_arg = emp\_arg;

}

}

@Jpf.FormBean

public static class RequestLoanFormBean implements java.io.Serializable {

private static final long serialVersionUID = 1L;

private int period\_arg;

private int minLoan\_arg;

public RequestLoanFormBean() {

}

public int getPeriod\_arg() {

return period\_arg;

}

public void setPeriod\_arg(int period\_arg) {

this.period\_arg = period\_arg;

}

public int getMinLoan\_arg() {

return minLoan\_arg;

}

public void setMinLoan\_arg(int minLoan\_arg) {

this.minLoan\_arg = minLoan\_arg;

}

}

}

### 3.2 index.jsp page

<%@ page language="java" contentType="text/html;charset=UTF-8"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui:html>

<head>

<netui:base/>

</head>

<netui:body>

<h1>\*\*\*\*\*Loan comparison\*\*\*\*\*</h1>

<img src="../money.jpg" width="250" height="90"/>

<h3>Get the best loan deals today from a well known lenders!!!!!!</h3>

<h4>Just fill the form bollow and click submit</h4>

<netui:form action="requestLoan">

<div>

<table>

<tr valign="top">

<td><netui:label for="actionForm\_minLoan\_arg" value="Amount of the loan £:" /></td>

<td><netui:textBox dataSource="actionForm.minLoan\_arg" tagId="actionForm\_minLoan\_arg"></netui:textBox></td>

</tr>

<tr valign="top">

<td><netui:label for="actionForm\_period\_arg" value="Period of the Loan in months:" /></td>

<td><netui:textBox dataSource="actionForm.period\_arg" tagId="actionForm\_period\_arg"></netui:textBox></td>

</tr>

</table>

</div>

<netui:button value="Submit" type="submit" />

</netui:form>

<h3>Would you like to check your Credit Rating in one Click?</h3>

<h4>Just enter your details bellow and click on submit</h4>

<netui:form action="checkCreditRating">

<div>

<table>

<tr valign="top">

<td><netui:label for="actionForm\_emp\_arg" value="Are you Employed?" /></td>

<td><netui:checkBox dataSource="actionForm.emp\_arg" tagId="actionForm\_emp\_arg"></netui:checkBox></td>

</tr>

<tr valign="top">

<td><netui:label for="actionForm\_ni\_arg" value="Natinal Insuranse Number:" /></td>

<td><netui:textBox dataSource="actionForm.ni\_arg" tagId="actionForm\_ni\_arg"></netui:textBox></td>

</tr>

<tr valign="top">

<td><netui:label for="actionForm\_salary\_arg" value="Anual Salary:" /></td>

<td><netui:textBox dataSource="actionForm.salary\_arg" tagId="actionForm\_salary\_arg"></netui:textBox></td>

</tr>

</table>

</div>

<netui:button value="Submit" type="submit" />

</netui:form>

</netui:body>

</netui:html>

### 3.3 loanDetailsPage.jsp page

<%@ page language="java" contentType="text/html;charset=UTF-8"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui-data:declarePageInput name="requestLoanResult" type="loanservice.Loan[]" required="true"/>

<netui:html>

<head>

<netui:base/>

</head>

<netui:body>

<h1>\*\*\*\*\*Loan comparison\*\*\*\*\*</h1>

<img src="../money.jpg" width="250" height="90"/>

<h2>This is your best Loan Quotes</h2>

<h3>Click on any quote to go direct to the lender web site</h3>

<table class="tablebody" border="1">

<tr class="tablehead" valign="top">

<th scope="col">Bank</th>

<th scope="col">MinLoan</th>

<th scope="col">MaxLoan</th>

<th scope="col">Period</th>

<th scope="col">Apr</th>

</tr>

<netui-data:repeater dataSource="pageInput.requestLoanResult">

<tr valign="top">

<td><netui:span value="${container.item.bank}" defaultValue="&nbsp;" /></td>

<td><netui:span value="${container.item.minLoan}" defaultValue="&nbsp;" /></td>

<td><netui:span value="${container.item.maxLoan}" defaultValue="&nbsp;" /></td>

<td><netui:span value="${container.item.period}" defaultValue="&nbsp;" /></td>

<td><netui:span value="${container.item.apr}" defaultValue="&nbsp;" /></td>

</tr>

</netui-data:repeater>

</table>

<p>Note: Apr is calculated in percentages %</p>

<a href="index.jsp">Back to Home Page</a>

</netui:body>

</netui:html>

### 3.4 creditRatingPage.jsp.page

<%@ page language="java" contentType="text/html;charset=UTF-8"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui-data:declarePageInput name="checkCreditRatingResult" type="java.lang.String" required="true"/>

<netui:html>

<head>

<netui:base/>

</head>

<netui:body>

<h1>\*\*\*\*\*Loan comparison\*\*\*\*\*</h1>

<img src="../money.jpg" width="250" height="90"/>

<h3>Your Credit Rating is:</h3>

<h2><netui:span value="${pageInput.checkCreditRatingResult}"></netui:span></h2>

<a href="index.jsp">Back to Home Page</a>

</netui:body>

</netui:html>