10-29-2010

Web Service Reliability for Deactivation and Decommissioning Knowledge Management Information Tool (D&D KM-IT) Vendor Management with Mobile Applications

Gowthami Thota
Florida International University, gthot001@fiu.edu

DOI: 10.25148/etd.FI10120603

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

WEB SERVICE RELIABILITY FOR DEACTIVATION AND DECOMMISSIONING KNOWLEDGE MANAGEMENT INFORMATION TOOL (D&D KM-IT) VENDOR MANAGEMENT WITH MOBILE APPLICATIONS

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Gowthami Thota

2010

To: Dean Amir Mirmiran
College of Engineering and Computing

This thesis, written by Gowthami Thota, and entitled Web Service Reliability for Deactivation and Decommissioning Knowledge Management Information Tool (D&D KM-IT) Vendor Management with Mobile Applications, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

_____
Leonel E Lagos

_____
Peter J. Clarke

_____
Ming Zhao, Major Professor

Date of Defense: October 29, 2010

The thesis of Gowthami Thota is approved.

_____
Dean Amir Mirmiran
College of Engineering and Computing

_____
Interim Dean Kevin O'Shea
University Graduate School

Florida International University, 2010

DEDICATION

I dedicate this dissertation to my parents NagaMalleswara Rao and Vijaya, and my siblings Chandu and Sirisha. Without their patience, understanding, support, blessings and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

I would like to thank members of my committee, Dr. Ming Zhao, Dr. Leonel E. Lagos and Dr. Peter J. Clarke for their help and support. I am heartily thankful to my mentor at the Applied Research Center (FIU-ARC), Mr. Himanshu Upadhyay, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. Lastly, I offer my regards to all of those who supported me in any respect during the completion of the project.

ABSTRACT OF THE THESIS

WEB SERVICE RELIABILITY FOR DEACTIVATION AND DECOMMISSIONING KNOWLEDGE MANAGEMENT INFORMATION TOOL (D&D KM-IT) VENDOR MANAGEMENT WITH MOBILE APPLICATIONS

by

Gowthami Thota

Florida International University, 2010

Miami, Florida

Professor Ming Zhao, Major Professor

This thesis presents the Knowledge Management Information Tool (KM-IT) Vendor Management Web Service and its reliability features. KM-IT is a web-based knowledge management information tool for the deactivation and decommissioning (D&D) user community. The Vendor Management (VM) module provides a directory service for searching D&D vendors. Clients have an increasing need to integrate and display the vendor information in their own applications. As such, this study proposes Web Services technology to provide VM access to clients. Moreover, when clients access information, particularly via mobile applications, they can encounter different failures that may occur on the network or the server. Such problems require fault tolerance in the VM Service. This study examines various reliability standards and selects the WS-ReliableMessaging standard as the best-suited approach to implement the reliability features for the VM Service. Implementation evaluation confirms that the KM-IT VM Service can effectively tolerate different types of failures.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

1. INTRODUCTION

Deactivation and Decommissioning Knowledge Management Information Tool (D&D KM-IT) is a web-based knowledge management information tool custom built for the D&D community. The system is currently being developed by the Applied Research Center (ARC) at Florida International University (FIU) in collaboration with the Department of Energy (U.S DOE), Energy Facility Contractors Group (EFCOG), and the As Low As Reasonably Achievable (ALARA) centers at Hanford and Savannah River Sites [23]. D&D KM-IT has various modules such as Hotline, Technology, Lessons Learned, and Vendor Management (VM). VM module provides a directory of D&D vendors along with their contact information.

Problem #1: Some clients, such as Oak Ridge National Laboratories (ORNL), working with DOE want to integrate their applications with KM-IT and display the vendor information on a regular basis, however the current technology and architecture of the D&D KM-IT is implemented as a tightly-coupled remote connection model. This means that for a client to use a method on a remote computer, both the client and the server have to have the appropriate libraries installed on their machines. So, integration between systems is a major problem.

Solution: Web Service is an emerging technology which can address this problem by providing a service-based model for developing distributed applications. Hence, exposing the application as a Web Service is the answer to the problems of interoperability and integration. Thus we proposed an idea to expose the Vendor Management module as a Web Service, so that it can be integrated into client applications.

Problem #2: When the client applications access information from the Web Service, they can encounter different failures from the network or the server. Such reliability problems make it important to include fault tolerance features in the KM-IT VM Web Service**.**

Solution: To overcome these failures, reliability was studied and its features were implemented for VM Web Service. Various reliability standards were researched and WebService-ReliableMessaging (WS-RM) standard was chosen for implementation.

Problem #3: The main challenge for the research was to test the service for reliability features. The Simple Object Access Protocol (SOAP) messages in the communication were intercepted in order to test the reliability features implemented for the service, thus paving the way for further research on various testing strategies.

Solution: The challenge of testing the service was addressed by conducting the experiments using a tool that intercepted messages. FIDDLER, a web debugging proxy that logs all Hypertext Transfer Protocol (HTTP) traffic between the computer and the internet, allows inspecting all HTTP(S) traffic, setting breakpoints, and "fiddling" with incoming and outgoing data. This is used to debug traffic from virtually any application, including Internet Explorer, Mozilla Firefox, Opera and others [3].

This thesis consists of six chapters, beginning with an introduction to the VM Web Service that was developed to integrate into the D&D KM-IT. I worked as part of a team consisting of my major professor, Dr. Ming Zhao, the project manager, Mr. Himanshu Upadhyay, and a graduate student, Ms. Harini Kondamudi. I was involved in developing the VM module and configuring its reliability. The purpose of VM Web Service is to provide vendor information to the user and ensure that the information is received even when the service is down for a certain period of time. This thesis describes the overall

application development process involved in creating a reliable VM Web Service, which includes development of the Web Service, hosting on the Internet Information Service (IIS), creation of a client, addition of reference of the service to the client located on different machines, load testing the web application, and finally conducting experiments to test the reliability features of the service. The first experiment is a best-case scenario to ensure that the application adheres to the standards of WS- ReliableMessaging. The second and the most important experiment, is the failure test involving requests from the client to a broken service and ensuring that the service receives and processes the request once it is up and running.

Chapter two provides the essential background material for this study and includes concepts such as Service Oriented Architecture, Web Service, Windows Communication Foundation, reliability and testing. This chapter also introduces VM Web Service and answers the questions of why D&D KMIT needs a reliable Web Service. The design of Web Service and its reliability features are presented in Chapter three while Chapter four describes the implementation of the research study, using Windows Communication Foundation (WCF). This includes creating the service and choosing a hosting mechanism (IIS) for the service. Chapter five presents the evaluation of the service using different client applications, unit testing and load testing of the application, which helps in application manageability and ensures functionality, and also documents the results of the experiments conducted. Finally, conclusions are presented in Chapter Six.

2. BACKGROUND

This chapter introduces some of the important definitions related to this research, starting from the simple definition of a service to the complex concept of WS-ReliableMessaging protocol. Even though later sections discuss the reliability in details, a brief overview is presented in this section.

2.1 Definitions

This chapter introduces basic concepts that are used throughout this thesis. The first section presents the basic understanding of Service Oriented Architecture (SOA). The second section introduces Web Services. Section three gives a basic description of all the terms in Windows Communication Foundation (WCF). Section four describes the features handled by the WS-ReliableMessaging protocol. Finally, the last section introduces testing terminology.

2.1.1 *Service Oriented Architecture (SOA)*

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed [4].

Services: Services are what you connect together using Web Services. A service is the endpoint of a connection. Also, a service has some type of underlying computer system that supports the connection offered [5].

Connections: Figure 1 gives a better understanding of connection between the client and service.

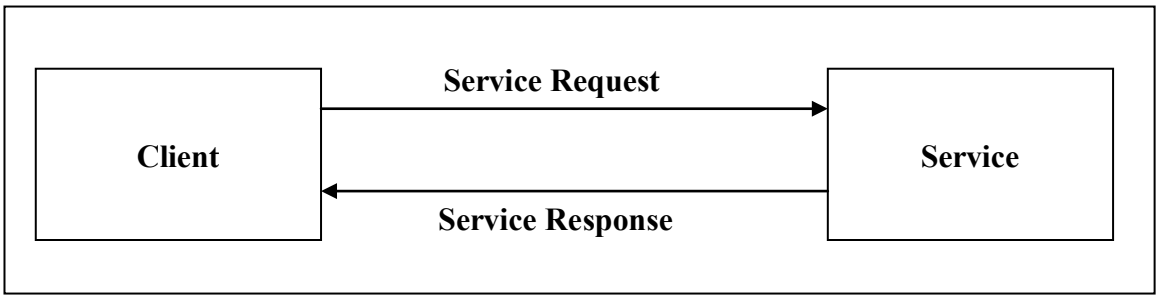


Figure 1: Illustration of Connections

The client sends a service request to the service. The service returns a response message to the client. The request and subsequent response connections are defined in some way that is understandable to both the service and client. A service provider can be a service consumer and vice-versa.

2.1.2 *Web Services*

Web Services (sometimes called application services) are services (usually including some combination of programming and data, but possibly including human resources as well) that are made available from a business web server for web users or other Web-connected programs. Providers of Web Services are generally known as application service providers [6]. Figure 2 illustrates the architecture of Web Services.

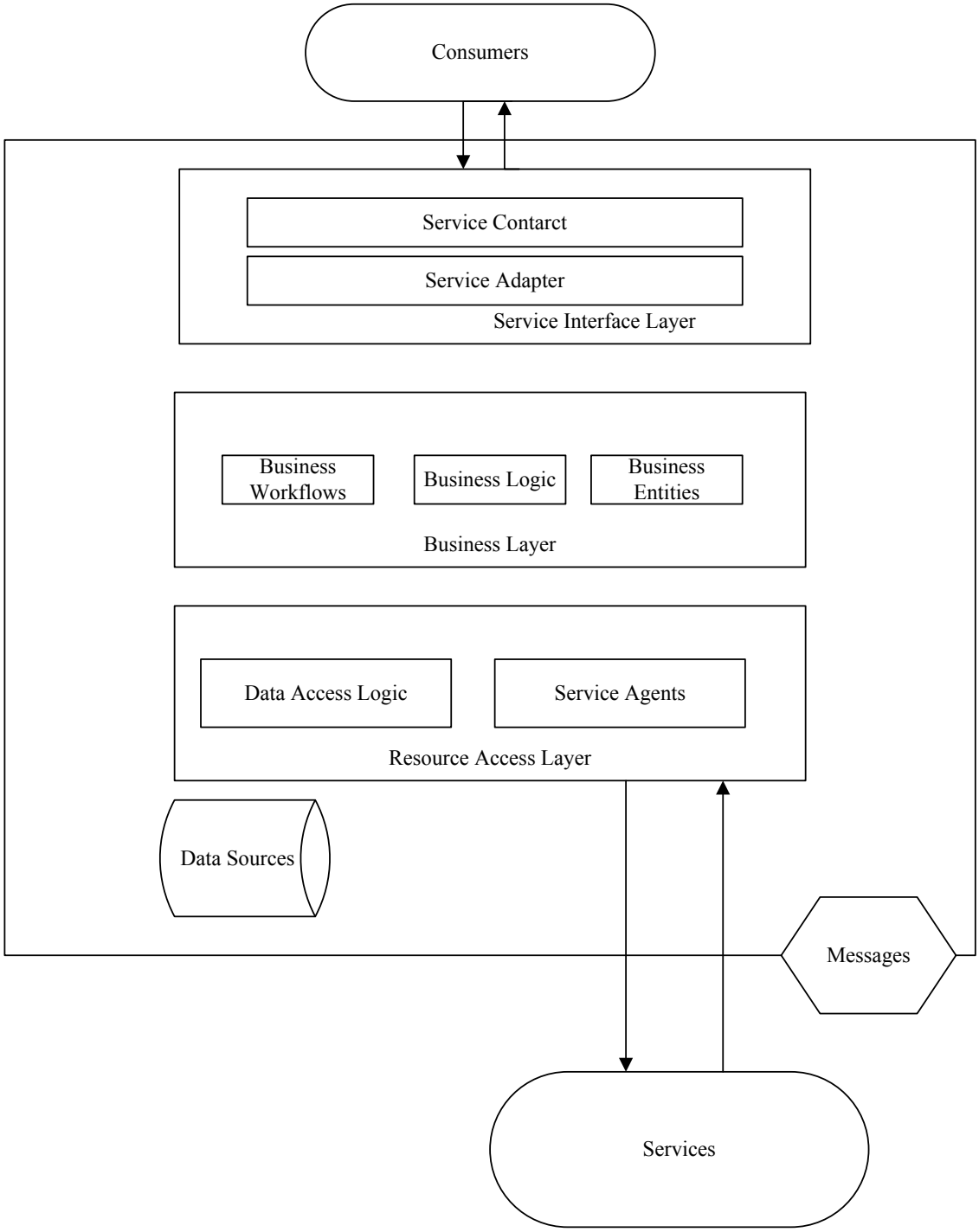


Figure 2: Web Service Architecture

Web Services Description Language (WSDL): WSDL forms the basis for Web Services; Figure 3 illustrates the use of WSDL.

The steps involved in providing and consuming a service is:

- A service provider describes its service using WSDL. This definition is published to a directory of services. The directory could use Universal Description, Discovery, and Integration (UDDI). Other forms of directories can also be used [7].

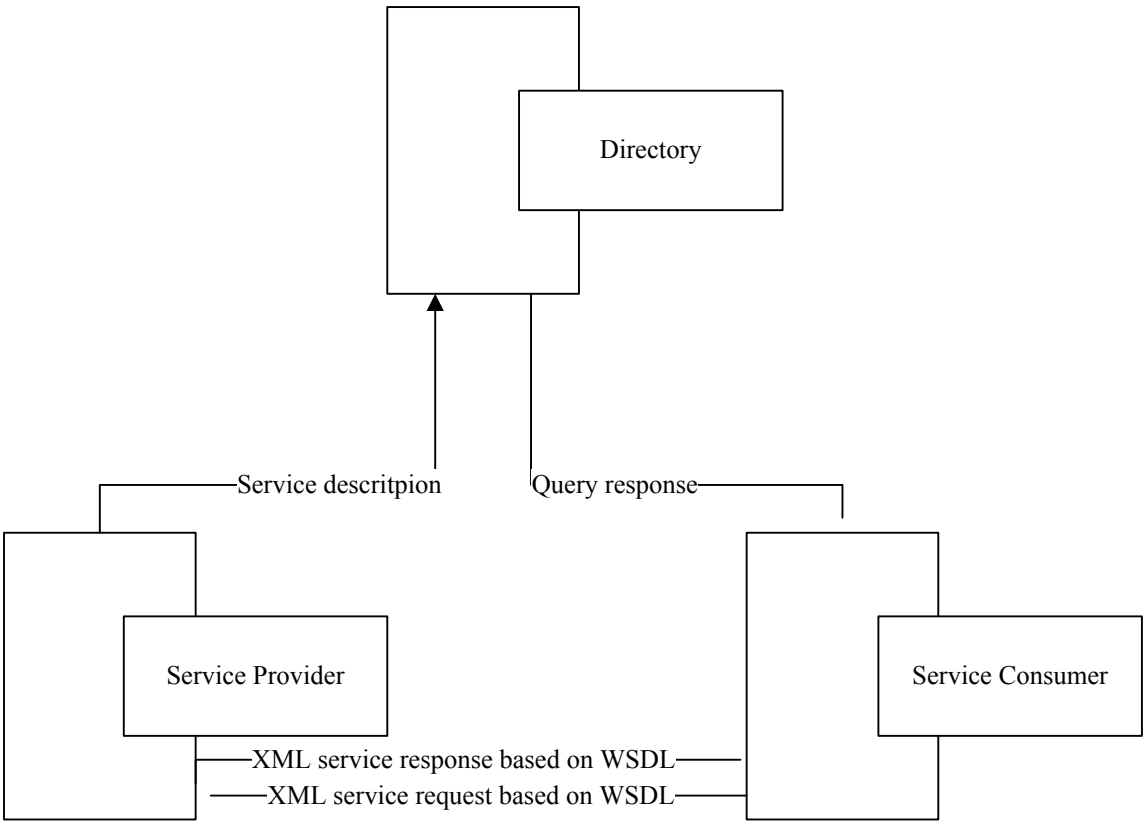


Figure 3: Illustration of WSDL

- A client issues one or more queries to the directory to locate a service and determine how to communicate with that service.

- A part of the WSDL provided by the service provider is passed to the client. This tells the client what the requests and responses are for the service provider.
- The client uses the WSDL to send a request to the service provider.
- The service provider provides the expected response to the client.

Universal Description, Discovery, and Integration (UDDI): UDDI is a public registry designed to house information about businesses and their services in a structured way. Through UDDI, one can publish and discover information about a business and its Web Services. This data can be classified using standard taxonomies so that information can be found based on categorization. Most importantly, UDDI contains information about the technical interfaces of a business service. Through a set of SOAP-based XML API calls, one can interact with UDDI at both design time and run time to discover technical data, such that those services can be invoked and used. In this way, UDDI serves as an infrastructure for a software landscape based on Web Services [8].

Simple Object Access Protocol (SOAP): SOAP provides a simple, extensible, and rich XML messaging framework for defining higher-level application protocols; offering increased interoperability in distributed and heterogeneous environments [9].

Extensible Markup Language (XML): A universal file format for storing and exchanging structured data. Like HTML, XML uses tags to define (or "mark up") the purpose of each piece of information in a file [10].

2.1.3 *Windows Communication Foundation (WCF)*

Windows Communication Foundation (WCF) is an application-programming interface in the .NET framework for building connected service-oriented applications.

Using WCF, one can send data as asynchronous messages from one service endpoint to another. A service endpoint can be part of a continuously available service hosted by IIS, or it can be a service hosted in an application [11].

Services: A WCF service is composed of three parts - a service class that implements the service, a host environment to host the service, and one or more endpoints to which clients will connect [12].

Addresses: In WCF, every service is associated with a unique address. The address provides two important elements: the location of the service and the transport protocol or transport schema used to communicate with the service. The location label of the address indicates the name of the target machine, site, or network; a communication port, pipe or queue; and an optional specific path or URL [13].

Bindings: A binding is merely a consistent, canned set of choices regarding the transport protocol, message encoding, communication pattern, reliability, security, transaction propagation and interoperability. Ideally, you would extract all these plumbing aspects out of your service code and allow the service to focus solely on the implementation of business logic [14].

Contracts: In WCF, all services expose contracts. The contract is a platform neutral and standard way of describing what the service does. WCF defines four types of contracts:

i. **Service Contracts**: Describes which operations the client can perform on the service [15].

ii. **Data Contracts**: Define which data types are passed to and from the service [15].

iii. **Fault Contracts**: Define which errors are raised by the service, and how the service handles and propagates errors to its clients [15].

iv. **Message Contracts**: Allow the service to interact directly with messages. Message contracts can be typed or un-typed, and are useful in interoperability cases and when there is an existing message format you have to comply with [15].

Endpoint: An endpoint is a referenceable entity, processor, or resource where Web Service messages originate or are targeted [16].

Features of WCF: One consequence of using WS standards is that WCF enables you to create service-oriented applications. The services have the advantage of being loosely-coupled instead of hard coded from one application to another. A loosely-coupled relationship implies that any client created on any platform can connect to any service as long as the essential contracts are met [17].

- **Interoperability:** WCF implements modern industry standards for Web Service interoperability.
- **Service Metadata:** WCF supports publishing service metadata using formats specified by industry standards such as WSDL, XML Schema and WS-Policy. This metadata can be used to automatically generate and configure clients for accessing WCF services.
- **Data Contracts:** WCF is built using the .NET Framework hence it includes code-friendly methods of supplying the contracts one wants to enforce. One of the universal types of contracts is the data contract.

- **Security:** Messages can be encrypted to protect privacy and one can require users to authenticate themselves before being allowed to receive messages. This security can be implemented using well-known standards such as SSL or WS-SecureConversation.
- **Multiple Transports and Encodings:** Messages can be sent on any of the several built-in transport protocols and encodings. The most common protocol and encoding are to send text encoded SOAP messages using the Hyper Text Transfer Protocol (HTTP) for use on the World Wide Web. Alternatively, WCF allows one to send messages over TCP, named pipes, or MSMQ. These messages can be encoded as text or using an optimized binary format. Binary data can be sent efficiently using the MTOM standard. If none of the provided transports or encodings suits ones needs, it is possible to create a custom transport or encoding.
- **Reliable and Queued Messages:** WCF supports reliable message exchange using reliable sessions implemented over WS- Reliable Messaging and using MSMQ.
- **Durable Messages:** A durable message is one that is never lost due to disruption in the communication. The messages in a durable message pattern are always saved to database. If a disruption occurs, the database allows one to resume the exchange when the connection is restored.
- **Transactions:** WCF also supports transactions using one of three transaction models: WS- Atomic Transactions, the APIs in the System.Transactions namespace and Microsoft Distributed Transaction Coordinator.
- **AJAX and REST Support:** REST is an example of an evolving Web 2.0 technology. WCF can be configured to process “plain” XML data that is not

wrapped in a SOAP envelope. WCF can be extended to support specific XML formats, such as ATOM and even non- XML format, such as JavaScript Object Notation (JSON).

- **Extensibility:** The WCF architecture has a number of extensibility points. If extra capability is required, there are a number of entry points that allow one to customize the behavior of a service.

2.1.4 *Reliability*

In WCF, reliability is controlled and configured in the binding. A particular binding can support or not support reliable messaging, and if supported can be enabled or disabled. Table 1 summarizes the relationship between binding, reliability, and ordered delivery and their respective default values.

Table 1: Relationship between Bindings and Reliability

Name	Supports Reliability	Default Reliability	Supports Ordered	Default Ordered
BasicHttpBinding	No	N/A	No	N/A
NetTcpBinding	Yes	Off	Yes	On
NetPeerTcpBinding	No	N/A	No	N/A
NetNamedPipeBinding	No	N/A	Yes	N/A
WSHttpBinding	Yes	Off	Yes	On
WSFederationHttpBinding	Yes	Off	Yes	On
WSDualHttpBinding	Yes	On	Yes	On
NetMsmqBinding	No	N/A	No	N/A

WSDualHttpBinding Class: A secure and interoperable binding that is designed for use with duplex service contracts that allows both services and clients to send and receive messages.

Namespace: System.ServiceModel

Assembly: System.ServiceModel (in System.ServiceModel.dll) [18]

WS-ReliableMessaging: WS-ReliableMessaging provides an interoperable protocol that a Reliable Messaging (RM) Source and Reliable Messaging Destination use to provide application source and destination; a guarantee that a message that is sent will be delivered. The guarantee is specified as a delivery assurance. The protocol supports the endpoints in providing these delivery assurances. It is the responsibility of the RM Source and RM Destination to fulfill the delivery assurances, or raise an error. The protocol allows endpoints to meet this guarantee for the delivery assurances [19]. Figure 4 illustrates the reliable messaging model.

There are four basic delivery assurances that the endpoints can provide:

- ***AtmostOnce:*** Messages will be delivered at most once without duplication or an error will be raised on at least one endpoint. It is possible that some messages in a sequence may not be delivered [19].
- ***AtLeastOnce:*** Every message sent will be delivered or an error will be raised on at least one endpoint. Some messages may be delivered more than once [19].
- ***ExactlyOnce:*** Every message sent will be delivered without duplication or an error will be raised on at least one endpoint. This delivery assurance is the logical "and" of the two prior delivery assurances [19].

- ***Inorder:*** Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions [19].

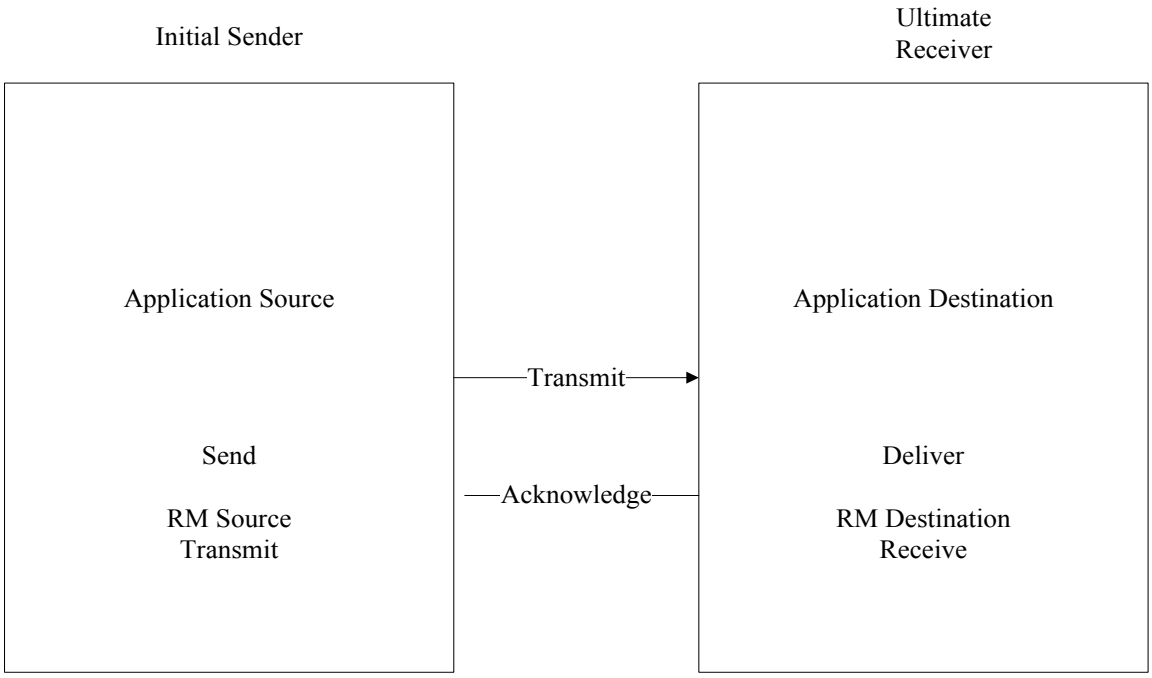


Figure 4: Reliable Messaging Model

2.1.5 *Testing*

Software testing is performed to verify that the completed software package functions according to the expectations defined by the requirements/specifications. The overall objective is not to find every software bug that exists, but to uncover situations that could negatively impact the customer, usability and/or maintainability [20].

In this thesis I conducted unit testing and load testing on the application, the definitions of which are given below:

Unit testing: The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as one expects. Each unit is tested separately before integrating it into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use [21].

Load Testing: Load testing is the process of placing a demand on a system or device and measuring its response. When the load placed on the system is raised beyond normal usage patterns in order to test the system's response at unusually high or peak loads, it is known as stress testing. The load is usually so high that the error conditions are the expected results, although no clear boundary exists of when an activity ceases to be a load test and becomes a stress test [22].

2.2 D&D KMIT

D&D KM-IT is a web-based knowledge management information tool custom built for the D&D community. The system is being developed by the Applied Research Center (ARC) at Florida International University (FIU) in collaboration with the U.S. Department of Energy (U.S DOE), EFCOG, and the ALARA centers at Hanford and Savannah River Sites [23]. Figure 5 shows the home page of the D&D KM-IT.

Figure 5: Home Page of D&D KM-IT

D&D KM-IT has various modules like Hotline, Technology, Lessons Learned, and VM module. Vendor Management module provides the vendor's information to the D&D users. Some clients such as OakRidge National Laboratories (ORNL) working with DOE want to integrate with KM-IT and display the vendor information in their own applications on a regular basis. To facilitate this module is exposed as Web Service, to be integrated in client applications. But, sometimes the client applications accessing information from the Vendor Management Web Service are often encountered with problems like service application failures and unavailability making them unreliable to communicate with. To overcome this problem the major issues to be considered are quality of service and continuous service delivery. The SOAP messages have to be sent to the receiving Web Service with guaranteed delivery.

There are several reasons why we use Web Services, the first and foremost is that Web Services operates on Service Based Architecture. In this Model, the Web Service provides a service and a client application can request and consume a service and terminate the request when it requires. Here the client application can be developed by any tool and it can run on any platform. The major advantage with Web Services is its interoperability. The other reasons for using Web Services include its ease of communication between different entities without affecting their existence, its support for application to application communication as the service is written separately from the application logic, the way they improve the information flow between applications, allows to communicate using the Extensible markup language, a text based protocol that all applications understand, reduce the licensing cost and that they do not rely on special protocols. Web Services are one way of reducing the cost of application integration for both internal system as well as business partners. They provide software reusability in distributed systems where applications execute across multiple computers on a network. So unless these Web Services are made reliable, organizations cannot trust them for critical operations such as complex business-to-business transactions or real time business integrations. Many clients of D&D KMIT want to take advantage of the above mentioned features provided by the Web Services. Therefore, we proposed the idea of developing Web Service for VM module.

There are several key things that are taken into consideration while developing the VM Web Service. For a service like VM which is to be used by many client applications, the service is designed in such a way that only the interface matters. The client can be on any platform and can use the service just by adding a reference to it. They have no idea of

how the service is implemented. They are just provided with the location of the service, what is its functionality, and how it can be used. The main focus while developing the Web Service should be given to the service contract and the data contract.

2.2.1 *Vendor Management Web Service Architecture*

The objective of VM Web Service is to provide reliable information to the mobile application and to ensure that there is guaranteed message delivery. Figure 6 gives the architecture of VM Web Service.

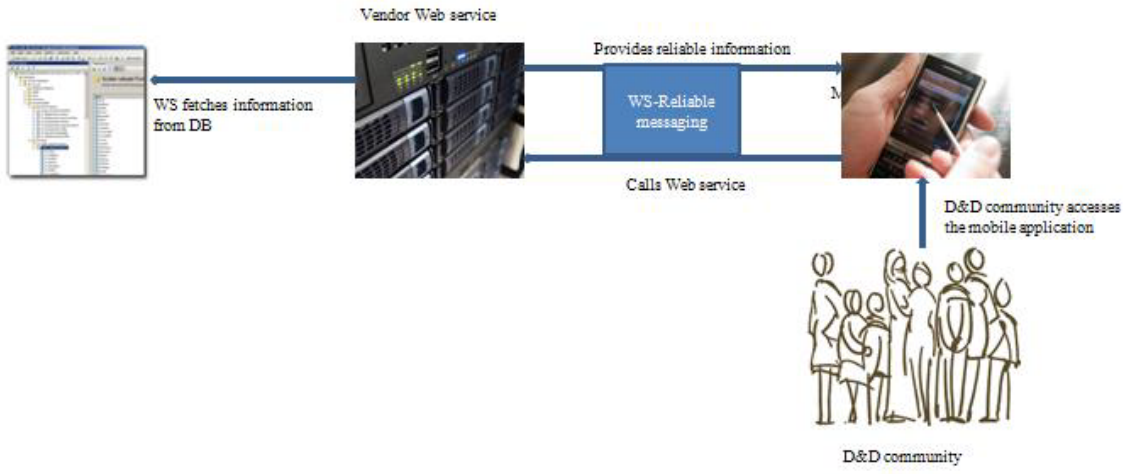


Figure 6: Communication Flow in Vendor Management Web Service

Whenever a D&D user wants to access the information of particular vendor through the mobile application, the mobile application calls the VM Web Service, and then the VM Web Service fetches the information from the database and transfers reliable information to the client. The underlying session between the client and the service is made reliable using WS-ReliableMessaging specification.

3. DESIGN

This section describes the available standards to implement reliability features. After the careful study about the standards, WS-RM was selected to be the appropriate reliability model that satisfies the scenario of Vendor Web Service. This section also describes WCF, a tool to implement WS-RM reliability standard.

3.1 Reliability Standards

The various standards to implement the reliability features include:

- **Service Broker:** Service Broker is used to add reliability to Web Service calls. The application that calls the Web Service can simply begin a service broker conversation and send a Web Service request over to a proxy service broker service that handles the task of performing the Web Service request and sends back the response in a reliable manner. The proxy broker service determines the transient errors and retrying the Web Service request.
- **WS-Reliability:** WS-Reliability is a generic and open model for ensuring reliable message delivery for Web Services. It defines a reliable message delivery as the ability to guarantee message delivery to software applications, Web Services or Web Service client applications with a chosen level of Quality of Service (QOS).
- **WS-Addressing:** Provide transport neutral mechanisms to address Web Services and messages. Specifically, this specification defines XML [XML 1.0, XML Namespaces] elements to identify Web Service endpoints and to secure end-to-end endpoint identification in messages.

- **WS-Bus:** Web Service Message Bus (WSBus) is a lightweight service-oriented middleware for reliable and fault tolerant Web Service interactions.
- **WS-Reliable Messaging:** This is an OASIS standard. WS-Reliable Messaging describes a protocol that allows message to be delivered reliably between distributed applications in the presence of software component, system, or network failure. The protocol is described in this specification in an independent manner allowing it to be implemented using different network transport technologies. To support interoperable Web Service, a SOAP binding is defined within this specification.

3.2 Standard Implemented for Vendor Management Web Service

Messages that are sent may never reach, arrive too late, arrive multiple times, or arrive out of order. After messages arrive, the service may crash and lose some messages [25]. Such problems are very difficult to deal with because the client is a Web Service instead of a human. Application code can handle these failures with an extra code, sometimes requiring the cooperation of the remote endpoints using an enhanced message exchange pattern. This can make the application expensive, very difficult and time-consuming to adapt to its changing requirements in future. As an alternative, reliable messaging (RM) can be used as a part of overall application failure recovery to improve productivity and time. Vendor Management Web Service is configured with reliability using WS-RM specification. The implementation details are provided in Chapter 4.

WS-RM Specification: To provide reliable messaging in Web Services, messages must be queued for the following reasons:

- For output messages, their transmission over the wire may need to be retried [25]
- For input messages, their delivery to the application may need to be retried, and their delivery order may not be same as their arrival order [25]

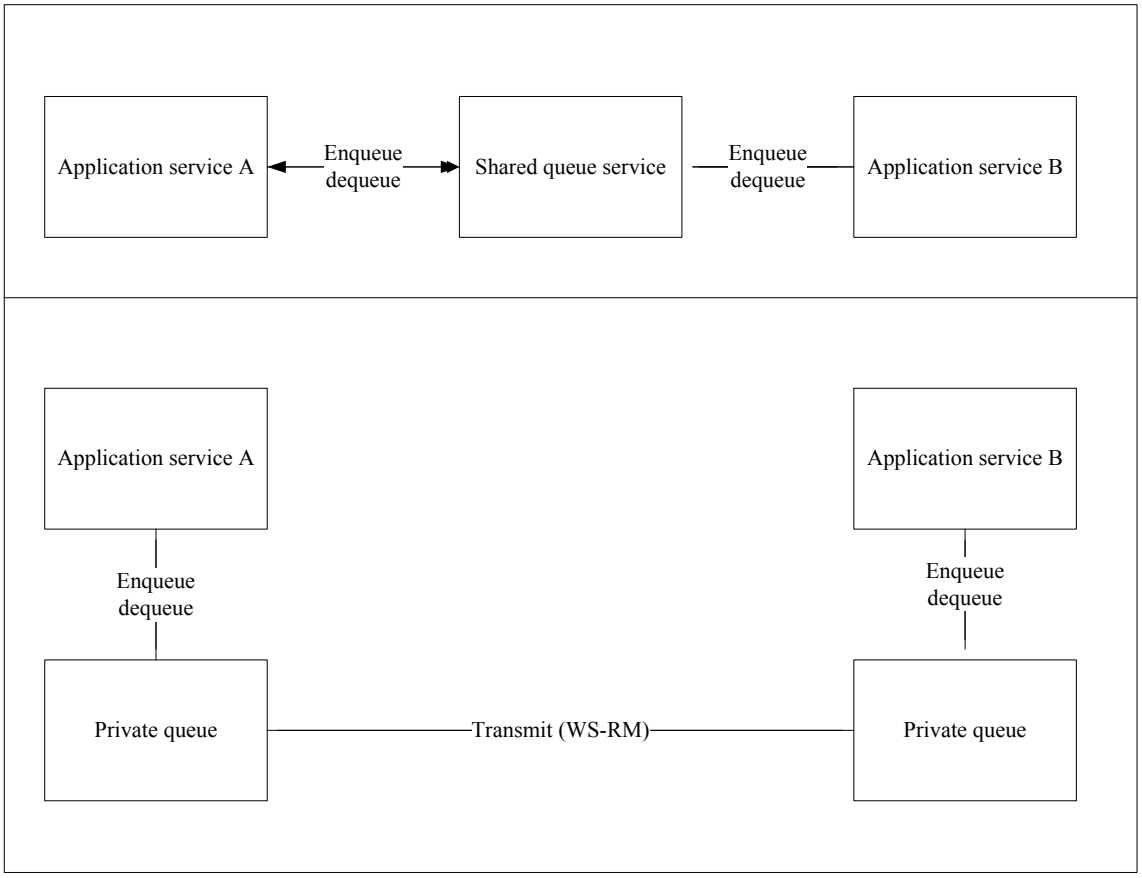


Figure 7: Role of WS-RM

WS-RM Model: Figure 8 illustrates the basic architecture and terminology used in the WS-RM specification.

1. The application client requests a sequence to be created with a **CreateSequence** message. If the application service accepts, it sends a **CreateSequenceResponse**. This establishes a sequence identifier for the sequence.

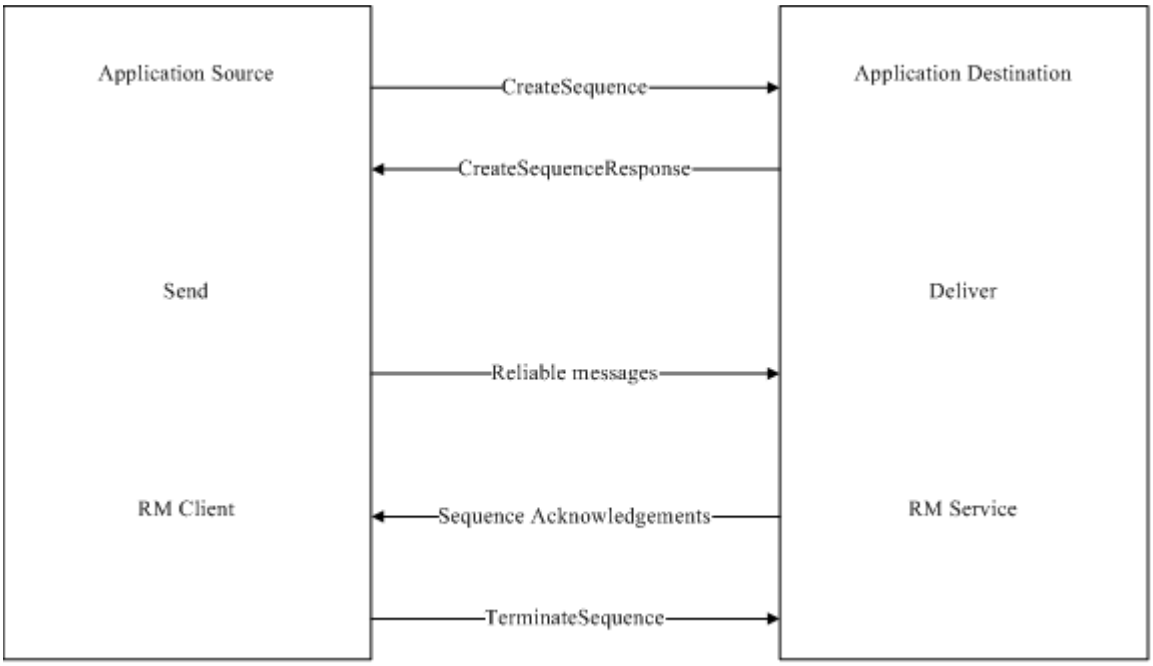


Figure 8: WS- RM Specification Model

2. The application client sends one or more messages with an RM header that contains the sequence identifier. This puts each message in the RM source's output queue.
3. After a message is in the output queue, the RM client transmits it over wire to the RM service. Within a sequence, transmitting can overlap with sending.
4. The RM service receives messages into its input queue.
5. The RM service acknowledges the messages received so far in that sequence.
6. The RM service delivers the message to the application client.
7. When the RM client receives the acknowledgements for the complete sequence, it sends a **TerminateSequence** message.

3.3 Tool To Implement Reliability Features for VM Web Service

Windows Communication Foundation (WCF) model unifies Microsoft's technologies for building distributed systems and incorporates a broader range of technologies and functionality. WCF is used to target the applications, which are developed using .NET framework 3.5 or higher. KM-IT is already developed in ASP.NET framework, so we plan to continue with the tool that gets along with the existing framework and technology. Vendor Management Web Service developed using WCF solves the problems for application integration and reliability.

Windows Communication Foundation (WCF): WCF is unified programming model for building service-oriented applications. SOA is an architectural design pattern by which several guiding principles determine the nature of the design. Basically, a SOA states that every component of a system should be a service, and that the system should be composed of several loosely coupled services. WCF is Microsoft's latest technology that enables applications in a distributed environment to communicate with each other. WCF is an umbrella technology that covers ASMX Web Services, .NET remoting, WSE, Enterprise service and System Messaging. It is designed to offer a manageable approach to distributed computing, broad interoperability, and direct support for service orientation. WCF supports many styles of distributed application development by providing a layered architecture. At its base, the WCF Channel architecture provides asynchronous, un-typed message-passing primitives. Built on top of this base are protocol facilities for secure, reliable, transacted data exchange and a broad choice of transport and encoding options. Figure 9 illustrates the major layers of the Windows Communication Foundation (WCF) architecture.

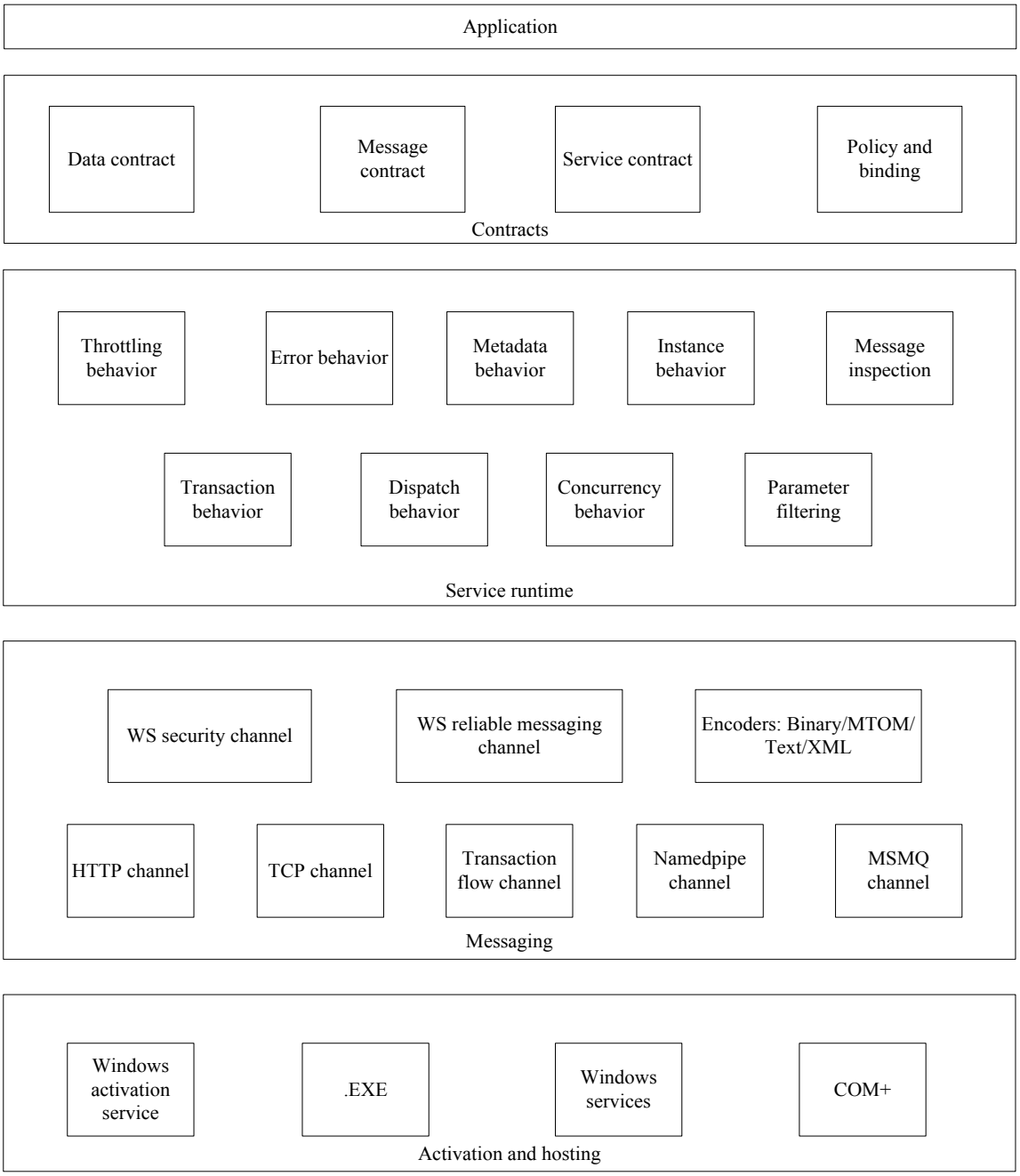


Figure 9: WCF Architecture

Contracts and Description: Contracts define different aspects of the message system. Policies and bindings stipulate the conditions required to communicate with a service. For example, binding must (at a minimum) specify the transport used (for example HTTP

or TCP) and an encoding. Policies include security requirements and other conditions that must be met to communicate with a service [26].

Service Runtime: The service runtime layer contains the behaviors that occur only during the actual operation of the service, that is, the runtime behaviors of the service [26].

Messaging: The messaging layer is composed of channels. A channel is a component that processes a message in some way. A set of channels is also known as channel stack. Channels operate on messages and message headers.

There are two types of channels: transport channels and protocol channels. Transport channels read and write messages from the network to and from the byte stream representation used by the network. Examples of transports are HTTP, named pipes, TCP and MSMQ. Protocol channels implement message-processing protocols, often by reading or writing additional headers to the message. Examples of such protocols include WS-Security and WS- Reliability. WS-Security is an implementation of the WS-Security specification enabling security at the message layer. The WS- Reliable Messaging channel enables the guaranteed message delivery [26].

Hosting and Activation: In its final form, a service is a program. Like other programs, a service must be run in an executable. This is known as self hosted service. Services can also be hosted, or run in an executable managed by an external agent such as IIS or Windows Activation Service (WAS) [26].

4. IMPLEMENTATION

4.1 Web Service for Vendor Management Module

This section describes the steps involved in creating the VM Web Service. There are two major steps involved in developing the Web Service:

Step1: Creating the service

Step2: Choosing the hosting mechanism

4.1.1 *Creating the Service*

- The first step to create the service is to open a new project in Visual Studio 2008.
- Name it **WcfServiceLibrary2,** as shown in Figure 10.

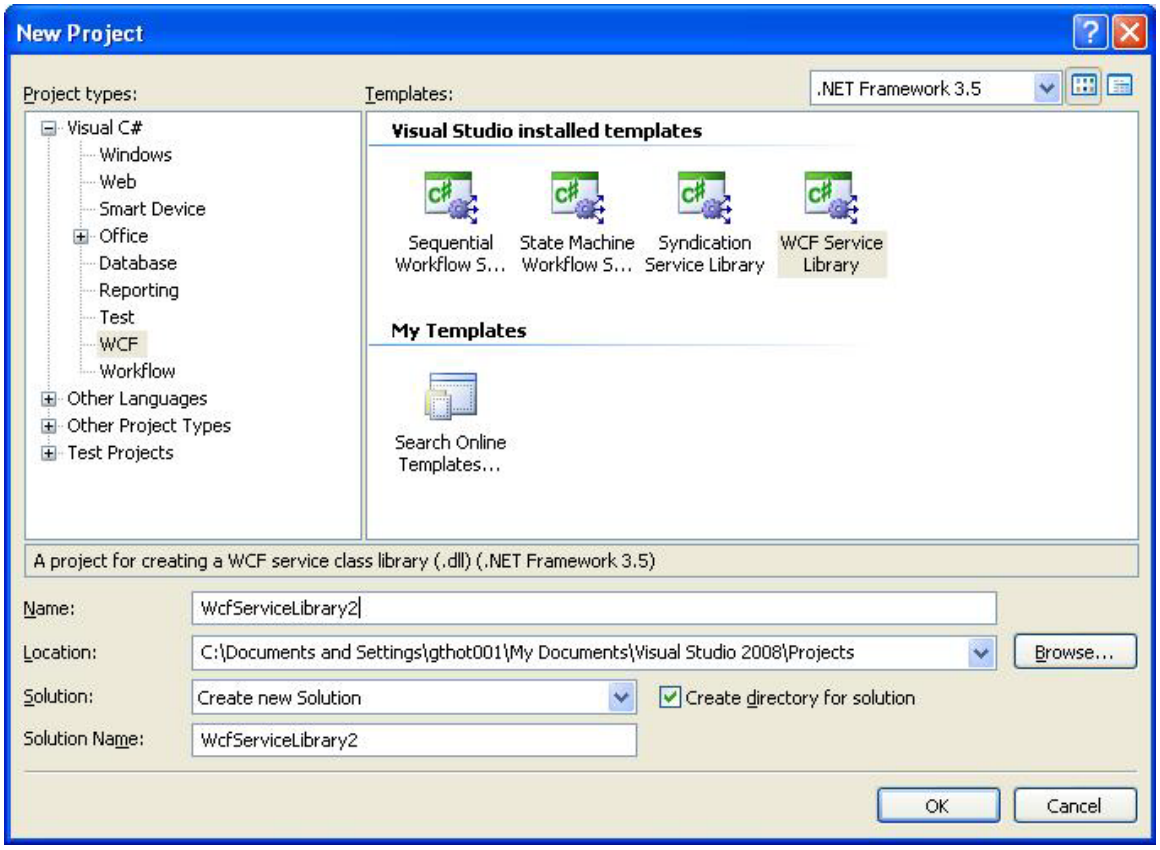


Figure 10: Naming the Project

- Specify the location as **C:\Documents and Settings\gthot001\My Documents\VisualStudio2008\Projects\WcfServiceLibrary2**.This is the location where the project will be saved.
- Once the project is created.
- Add a new WCF service in the Solution Explorer of the **WcfServiceLibrary2** project
- Name it **WcfServiceLibrary2,** as shown in Figure 11.

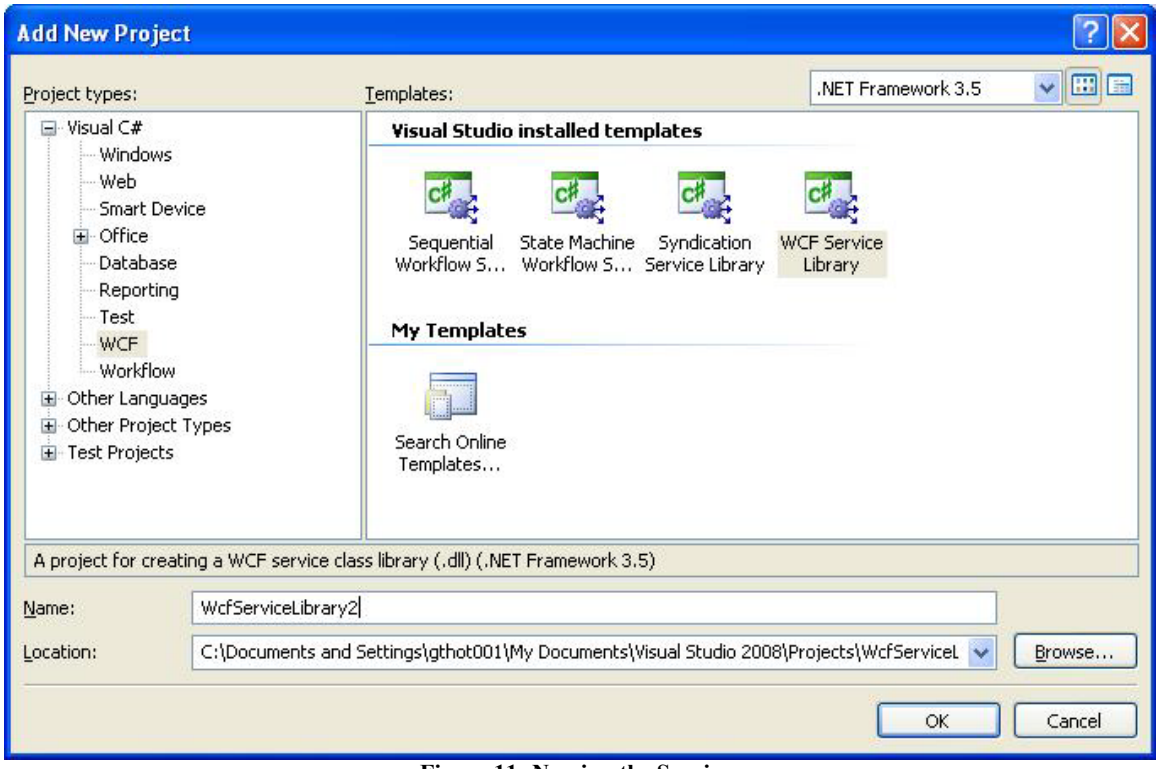


Figure 11: Naming the Service

- Specify the location as **C:\Documents and Settings\gthot001\MyDocuments\VisualStudio2008\Projects\WcfServiceLibrary2\WcfServiceLibrary2**. This is the location where the service is saved in the project.

- Once the service is created, it opens up **IService1.cs, Service1.cs,** and an application configuration file (Figure 12).

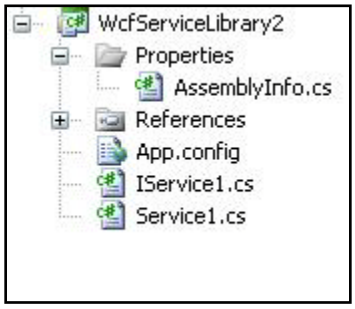


Figure 12: Service Created

- **IService1.cs** is a class that contains the service contract, which contains all the available operations (Figure 13).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using System.Data;

namespace WcfServiceLibrary2
{
    // NOTE: If you change the interface name "IService1" here, you must also update the reference to "IService1"
    [ServiceContract(SessionMode = SessionMode.Required)]
    public interface IService1
    {
        [OperationContract]
        DataSet vendorwebservice(string Vendor_name);
    }
}
```

Figure 13: Service Contract

- **Service1.cs** is a class that contains the actual implementation of the VM Service.
- It contains a **vendorwebservice** method; this takes the **Vendor_name** as the parameter, opens a connection to the database, fetches the information matching with that **Vendor_name**, fills that dataset and returns the dataset to the client to be displayed.

4.1.2 *Choosing the Hosting Mechanism*

Now that the WCF service is created, the next step is to consume it and to find an option to host the service. The first choice is creating an application and hosting the service within this application. However, this option is good only when we want to host the service quickly and test it. In a practical scenario, the service should reside at some place and it should be available to everyone at all times. Fortunately, WCF provides various options when it comes to hosting a service. These options are:

- Console application
- Winform application
- Internet Information Services (IIS)
- Windows service
- Windows Activation Service

The choice of hosting restricts the type of transport that we can use for that service. Hosting the service in Internet Information Services (IIS) best suits this application.

What is IIS? IIS is a World Wide Web Server, an FTP server all rolled into one. IIS means that we can publish WWW pages and extend into the realm of ASP (Active Server Pages) whereby JAVA or VBscript (Server side scripts) can generate pages on the fly. IIS has things like application development environment (FrontPage), integrated full-text searching (Index Server), multimedia streaming (NetShow), and site management extensions [27].

A WCF service that runs in the IIS environment takes full advantage of IIS features, such as process recycling, idle shutdown, process health monitoring, and message-based activation. This hosting option requires that IIS be properly installed and configured, but does not require that any hosting code be written as a part of the application. IIS hosting is used only with an HTTP transport.

This section describes the steps involved in hosting the application in an IIS. First, the service is configured to be hosted within ASP.NET web application, and then it is hosted in IIS.

Step 1: Add a **new website** to the solution, name the site **ServiceHost,** and give a location to the site, as shown in Figure 14.

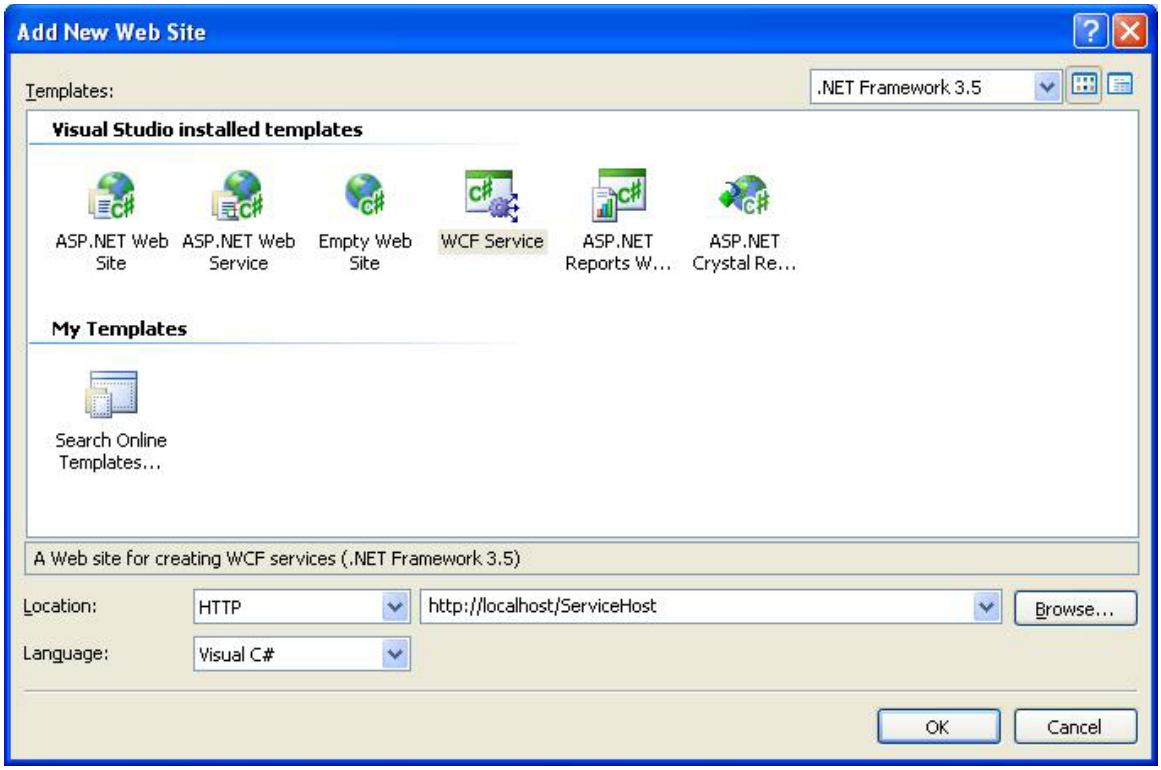


Figure 14: Creating a New Website to Host the Service

- The project template (Figure 15) gives a sample implementation of WCF service; since there is service implementation already, this is deleted.

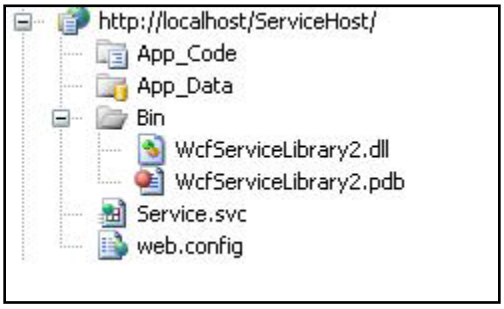


Figure 15: Project Template for the Hosted Site

Step 2: Right click on **http://localhost/ServiceHost/**, select **Add Reference** and select the WcfServiceLibrary2 (Figure 16).

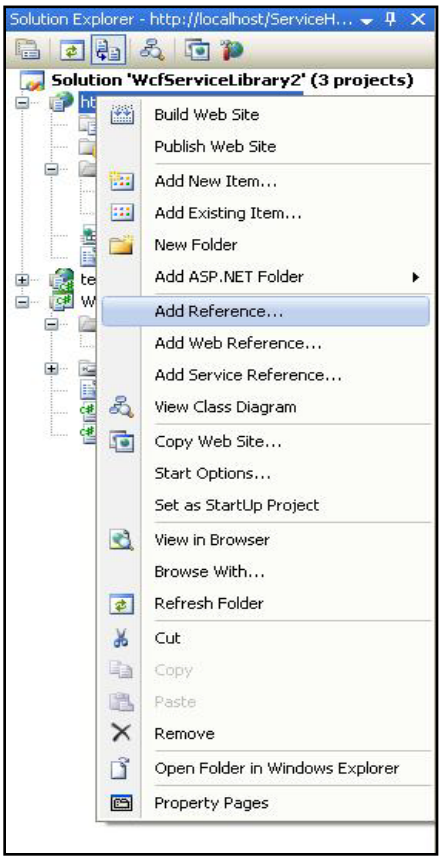


Figure 16: Adding a Reference to the Website

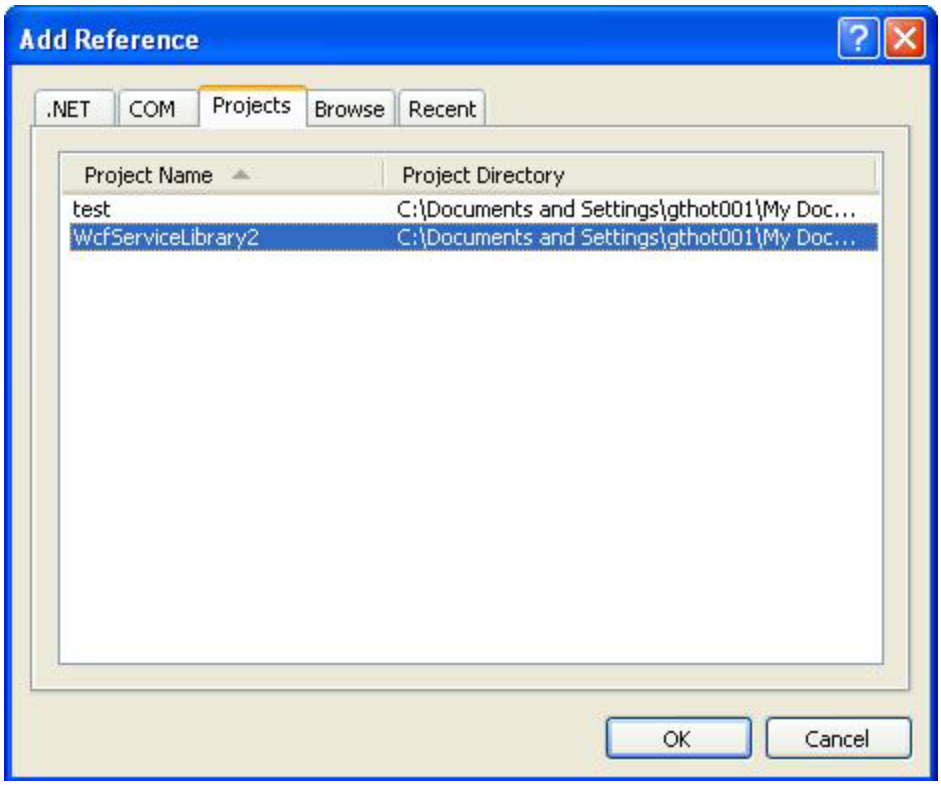


Figure 17: Adding the Reference

Step 3: Configure the svc file (Figure 18) that maps to the service to host the service inside the ASP.NET web application.

```
<%@ ServiceHost Language="C#" Debug="true" Service="WcfServiceLibrary2.Service1"  %>
```

Figure 18: Svc File Mapping to the Service

Specify to ASP.NET, information on how to map requests for Service.svc file to Service1.cs, which is in the service library, **WcfServiceLibrary2.**

Step 4: Configure the website with some endpoints for the service. When the service is browsed, ASP.NET development server opens up and it browses the Service.svc. A documentation page can be seen that is provided by WCF (Figure 19). The actual metadata exchange of the service and built-in metadata behavior is a part of the WCF.

Figure 19: Browsing the Service in an ASP.NET Environment

Step 5: The next step is to browse this service in IIS. First, launch the IIS management tool, as shown in Figure 20.

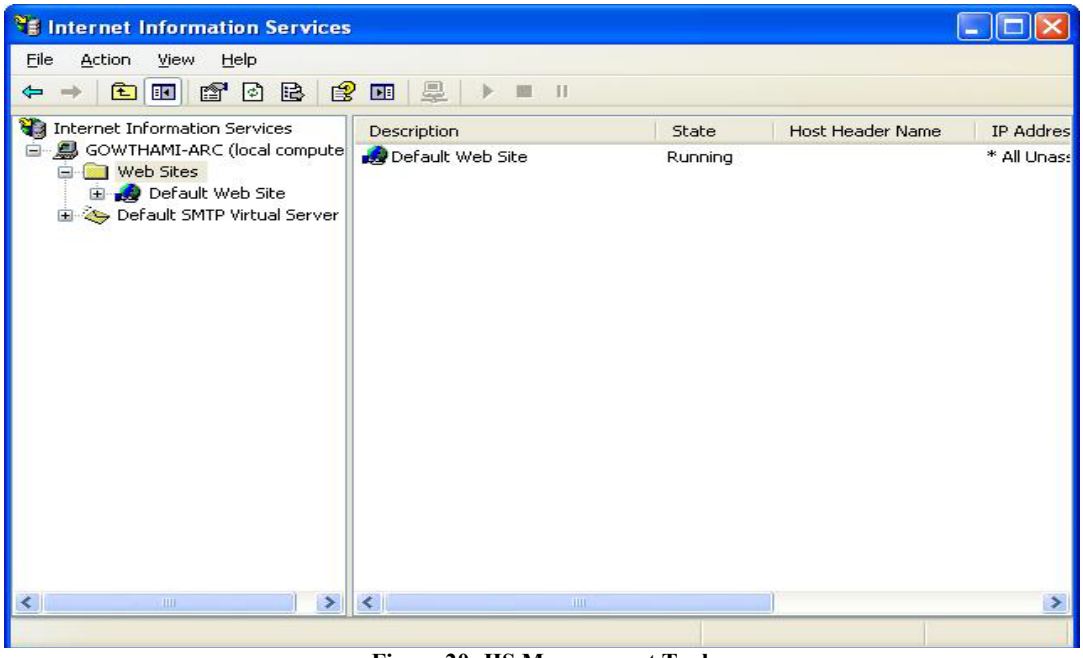


Figure 20: IIS Management Tool

- In the **Internet Information Services,** expand **Web Sites.**
- Expand **Default Web Site.**

- Select the **ServiceHost,** which is the virtual directory created.
- The content view of the directory holds the actual Service.svc, configuration files and App_code (Figure 21).

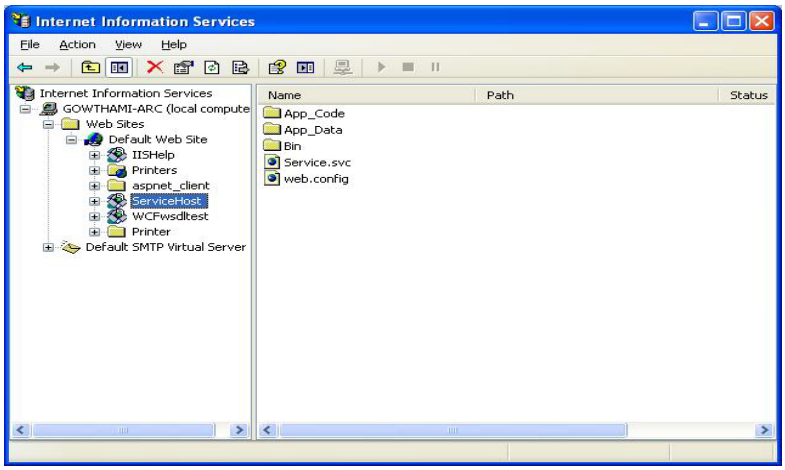


Figure 21: Content View

- Right click on the Service.svc and select **Browse**; this launches the service on the localhost (Figure 22).

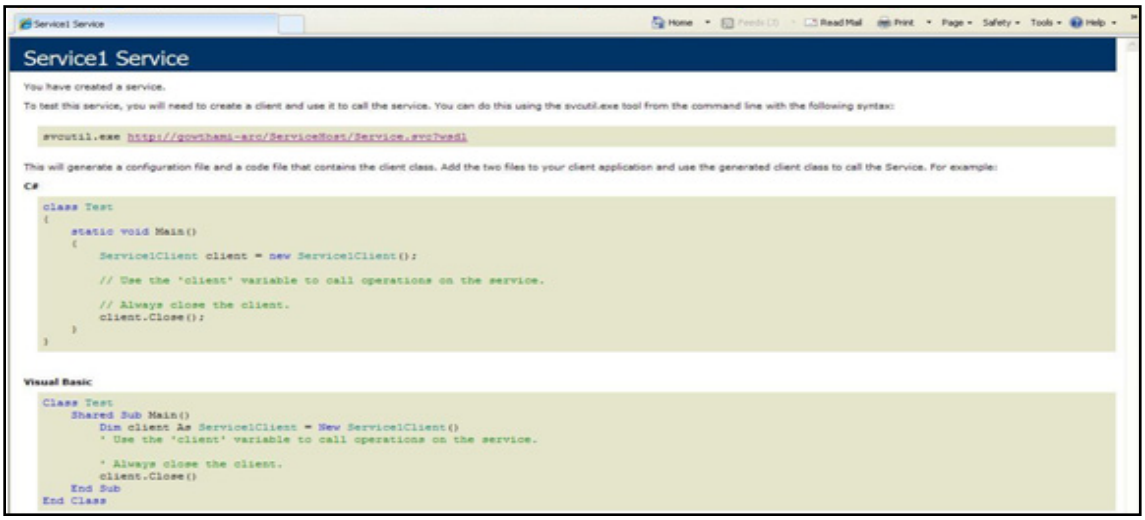


Figure 22: Browsing the Service in IIS

4.2 WS-RM Model for Vendor Management Web Service

The pillar in Windows Communication Foundation (WCF) that supports software-as-a-service is reliability, with which we can achieve the secure, transactable, and reliable services that WCF promises. This section briefly describes the implementation of the WS-RM model for VM Web Service by setting the configuration on the service side.

4.2.1 *Settings on the Service Side*

In order to achieve reliability features on the service side, the first step is to implement reliability features in WCF. The second step is to configure the service for reliability.

4.2.1.1 Implementing Reliability Features in WCF

WCF uses SOAP reliable messaging to provide end-to-end message transfer reliably between service endpoints in the system, and uses WS-Reliability to provide a layer of reliability between potentially unreliable or intermittently-connected networks. A number of different options are available for defining the specific reliability characteristics, such as exactly-once messaging, wherein a message is guaranteed to be delivered once and once only. When using WCF SOAP reliable messaging, the reliability protocol is end-to-end, regardless of the number of intermediaries between the endpoints. This is very important because in real world systems, we have a number of intermediaries between systems, such as HTTP proxies.

The service which is created earlier will not run because the web.config file is not configured properly. To get the service up and running, the following changes should be made:

- Change the code in the Service1.cs file (Figure 23) for implementing the service.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Data;
using System.Data.SqlClient;
using System.Text;

namespace WcfServiceLibrary2
{
    // NOTE: If you change the class name "Service1" here, you must also update the reference to "Service1" in App
    public class Service1 : IService1
    {
        #region IService1 Members

        [OperationBehavior(TransactionScopeRequired = true, TransactionAutoComplete = false)]
        public DataSet vendorwebservice(string Vendor_name)
        {
            DataSet dt = new DataSet();

            try
            {
                //Data Source = myServerAddress; Initial Catalog = myDataBase; User Id = myUsername; Password = my

                SqlConnection con = new SqlConnection(@"Data Source= GOWTHAMI-ARC\SQLEXPRESS;Initial Catalog= KMIT

                SqlCommand command = new SqlCommand("usp_vendorinformation", con);
                SqlCommand cmd = new SqlCommand();
                command.CommandType = CommandType.StoredProcedure;
                cmd.Connection = con;
                con.Open();

                command.Parameters.Add("@Vendor_name", SqlDbType.NVarChar).Value = Vendor_name;
                SqlDataAdapter adapter = new SqlDataAdapter(command);

                adapter.Fill(dt);

                return (DataSet)dt;
            }
            catch (Exception)
            {

                return (DataSet)dt;
            }
        }
```

Figure 23: Service1.cs File

- Change the Service.svc file to point to **WcfServiceLibrary2.Service1.**

```
<%@ ServiceHost Language="C#" Debug="true" Service="WcfServiceLibrary2.Service1"  %>
```

This attribute maps the service to the **Service1** class shown above, which is in **WcfServiceLibrary2** namespace.

- Finally, edit the app.config file (Figure 24) to initialize the System.ServiceModel that is the heart of WCF.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <!-- When deploying the service library project, the content of the config file must be added to the host's
  app.config file. System.Configuration does not support config files for libraries. -->
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="NewBinding0">
          <reliableSession enabled="true" />
          <security>
            <message clientCredentialType="Windows" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <services>
      <service behaviorConfiguration="WcfServiceLibrary2.Service1Behavior"
        name="WcfServiceLibrary2.Service1">
        <endpoint address="" binding="wsHttpBinding" bindingConfiguration="NewBinding0"
          contract="WcfServiceLibrary2.IService1">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8731/Design_Time_Addresses/WcfServiceLibrary2/Service1/" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="WcfServiceLibrary2.Service1Behavior">
          <!-- To avoid disclosing metadata information,
          set the value below to false and remove the metadata endpoint above before deployment -->
          <serviceMetadata httpGetEnabled="True"/>
          <!-- To receive exception details in faults for debugging purposes,
```

Figure 24: App.Config File

4.2.1.2 Configuring the Service for Reliability

Now that the service is up and running, the next step is to configure the service with reliability. There are various standard bindings that can be used in WCF to ensure reliability in the service. The system provides bindings that support reliable sessions, such as:

- WSDualHttpBinding
- WSHttpBinding
- WSFederationHttpBinding
- NetTcpBinding

This application uses WSHttpBinding, as it is secure, interoperable binding that is designed for use with duplex service contracts that allows both services and clients to send and receive messages.

Setting the bindings in the configuration file

To configure the service with reliability, an endpoint is created for the service. "Address," "Binding" and "Configuration" for the endpoint are configured in order to implement the reliable sessions.

- In the Solution Explorer, right-click the 'App.config' of the WCF service (WcfServiceLibrary2).
- Choose the **Edit WCF Configuration** option (Figure 25). If you do not see the **Edit WCF Configuration** option, click the **Tools** menu and select **WCF Service Configuration Editor**.
- Close the **WCF Configuration Editor Tool** that appears.
- The option should now appear on the 'App.config' context menu.

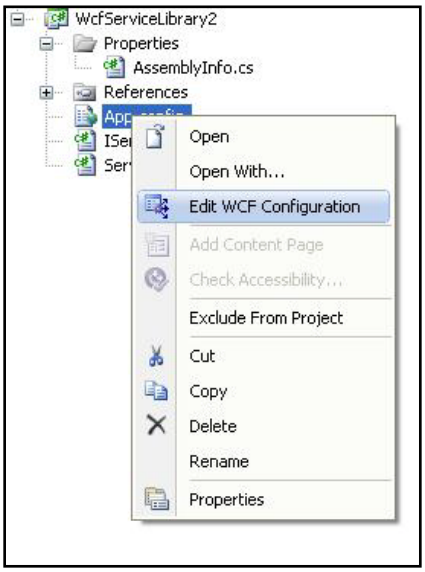


Figure 25: Edit Configuration

- Within the Configuration Editor, in the **Service** section, select **Create a New Service Endpoint.**
- When new Service Endpoint Element Wizard pops up, specify the service contract for the service as **WcfServiceLibrary2.IService1,** and click **Next** (Figure 26).

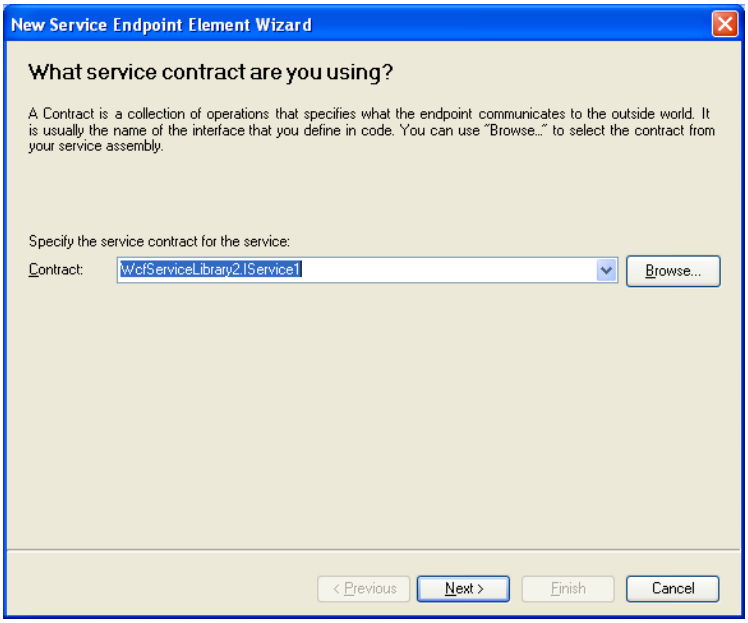


Figure 26: Specify Service Contract

- Check **New Binding Configuration** to create a new binding configuration for the service, and click **Next** (Figure 27).

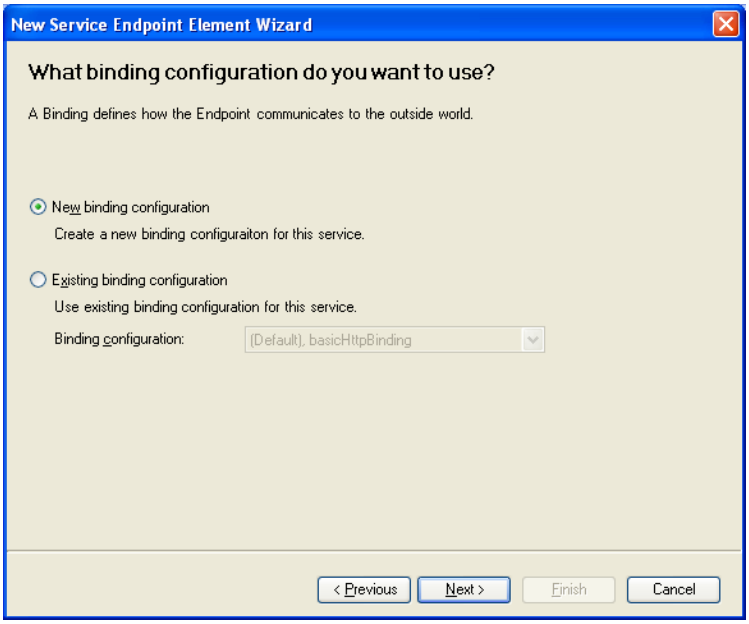


Figure 27: Binding Configuration for the Service

- Check **Advanced Web Service interoperability** to specify the method of interoperability for the service, and click **NEXT** (Figure 28).

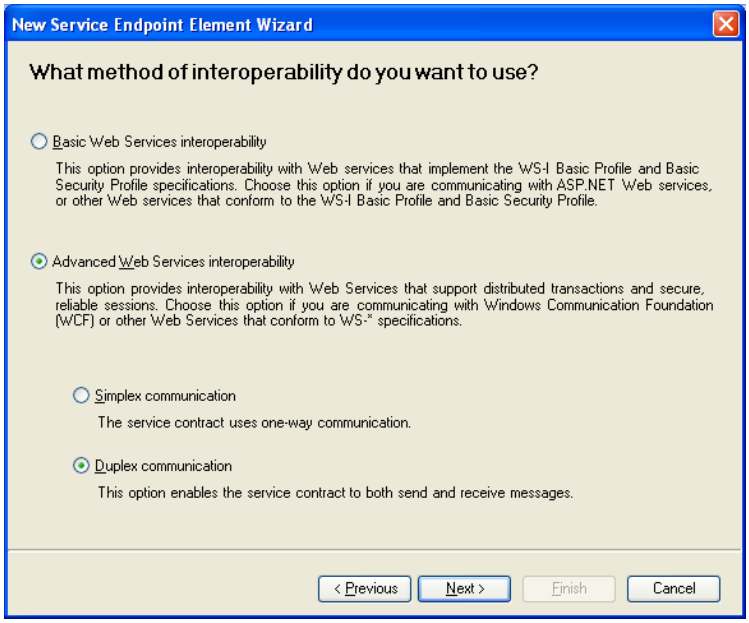


Figure 28: Communication Mode of the Service

- A new service endpoint configuration is created (Figure 29).

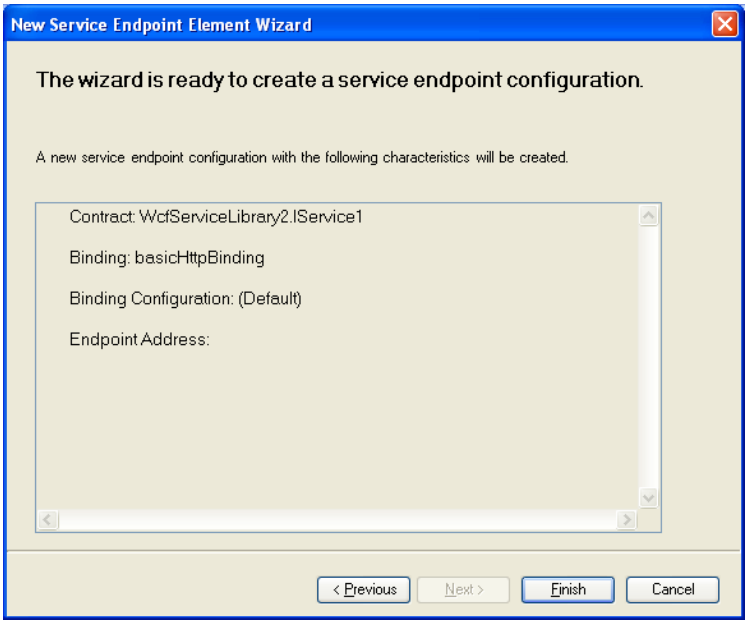


Figure 29: Service Endpoint

- Expand **Bindings** section in the **Configuration** section and click **New Binding Configuration** (Figure 30).

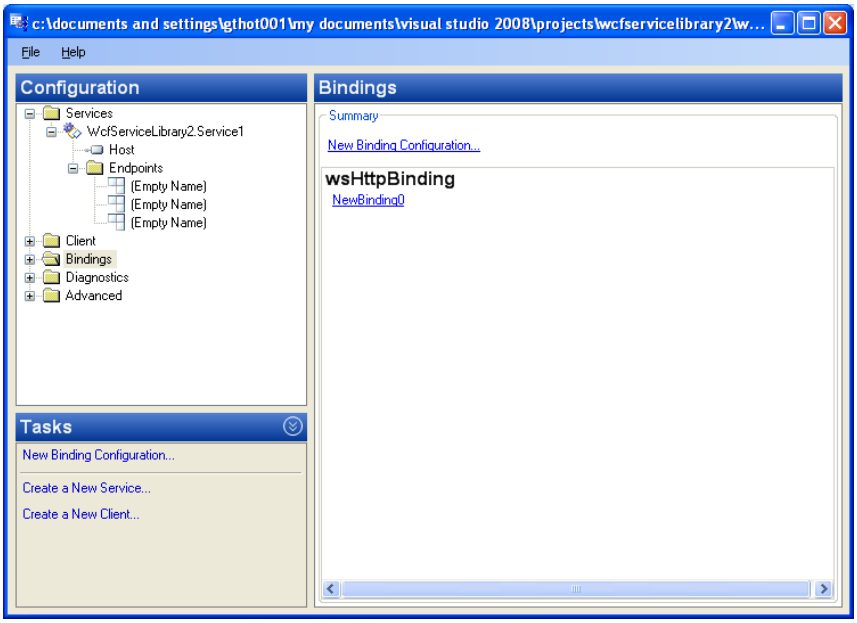


Figure 30: New Binding Configuration

- When **create new binding** window pops up, select wsHttpBinding and click **Ok** (Figure 31).

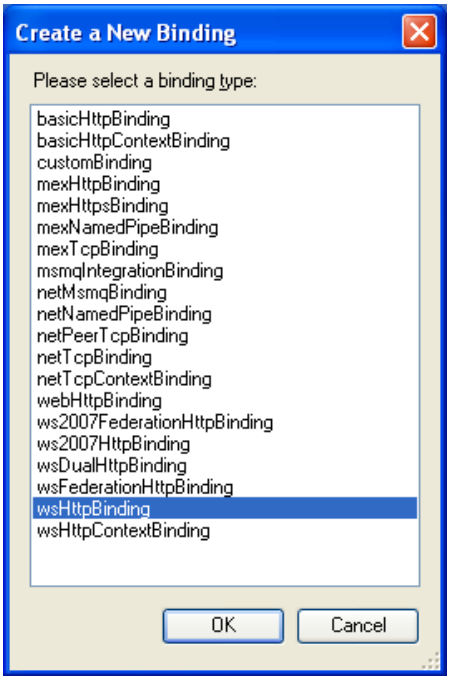


Figure 31: Create a New Binding

- Select the created binding, in the **Bindings** tab of **wsHttpBinding**, specifying the **ReliableSession** properties (Figure 32).

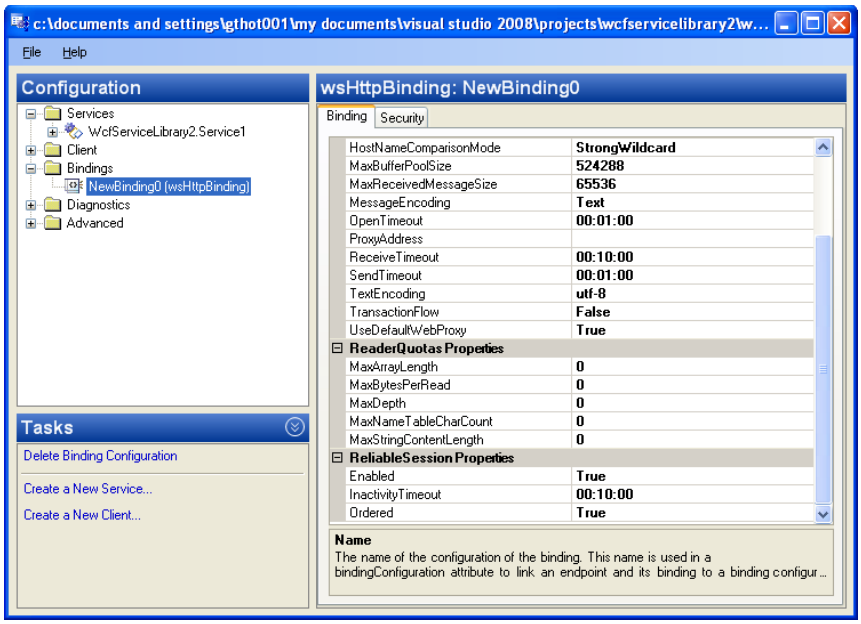


Figure 32: Enabling Reliable Session

- Select the created endpoint, in the **Service Endpoint** section, and **Endpoint properties** are specified, as shown in (Figure 33).

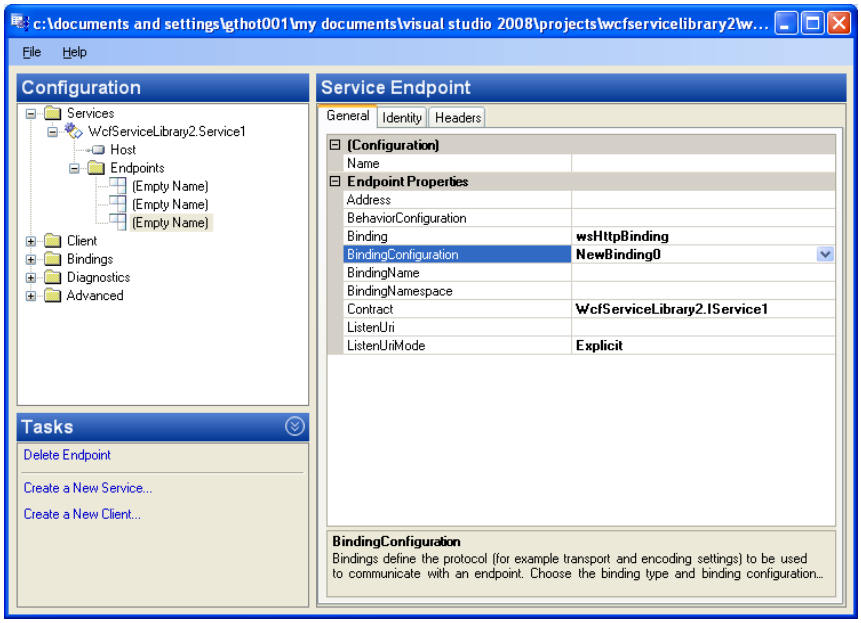


Figure 33: Configuring Endpoint

This describes the implementation of WS-RM model in WCF for the VM Web Service, as discussed in Chapter 3.

5. EVALUATION

5.1 Clients to Access Vendor Management Web Service

WCF is very symmetric. Most of the things that we already know, like addresses, bindings and service contracts apply to the clients in the same way. In fact, implementing a service client is quite often mostly a matter of using the right tool, just as with ASP.NET Web Services. For this project 3 different client applications are generated:

- Web Application
- Windows Forms Application
- Mobile Web Application

5.1.1 *Creating Web Application*

- Right click in the solution explorer.
- Expand **Add.**
- Select **New Project**.
- This pops up the **Add New Project** window.
- Expand the **Visual C#** in the **Web templates.**
- Select **ASP.NET Web Application**.
- Name it as **WebClient.**
- Give the location of the project as **C:\DocumentsandSettings\harini\MyDocuments\Visualstudio2008\Projects\ WcfClientLibrary2\WebClient** and
- Then hit **OK** (Figure 34).

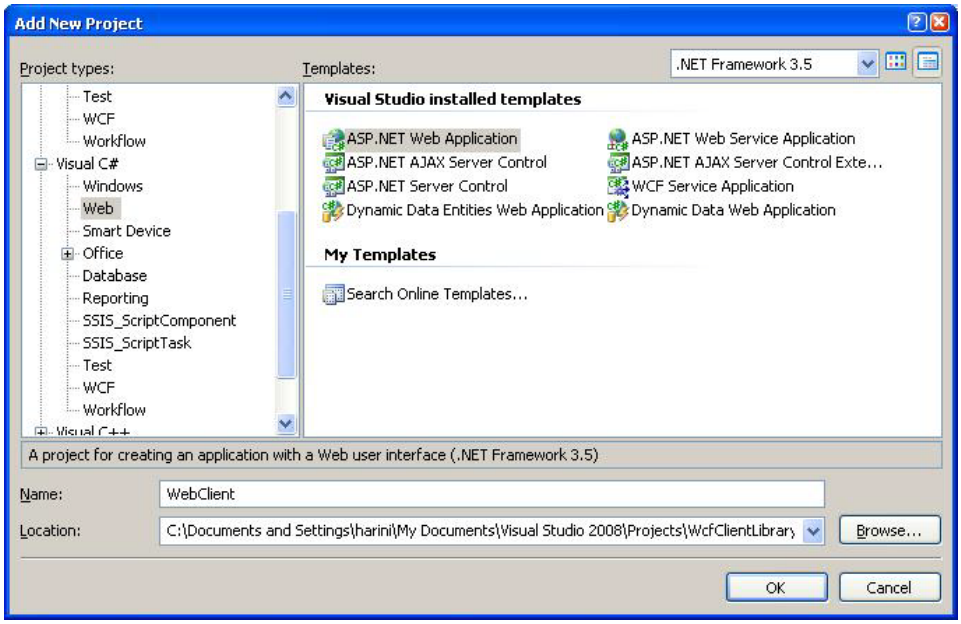


Figure 34: Creating a Web Client Application

- The project template provides a designer and code behind files (Figure 35).

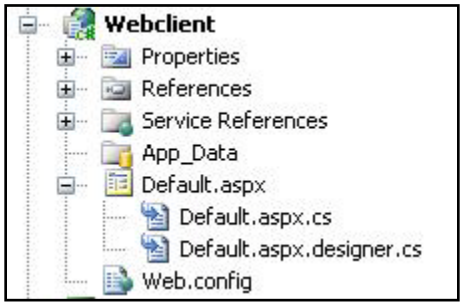


Figure 35: Web Client Project Template

- Next, generate a service reference that knows how to communicate with the service by right clicking on the references in the **WebClient** project, as shown in Figure 36.

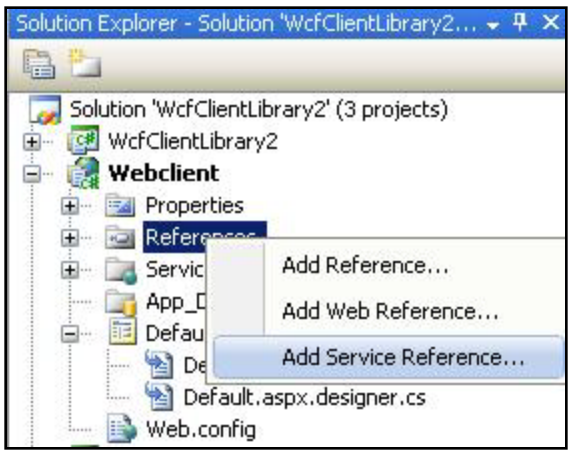


Figure 36: Adding Service Reference

- Before adding the service reference, launch the service library.
- Once the service is launched, copy the address to the clipboard (Figure 37).

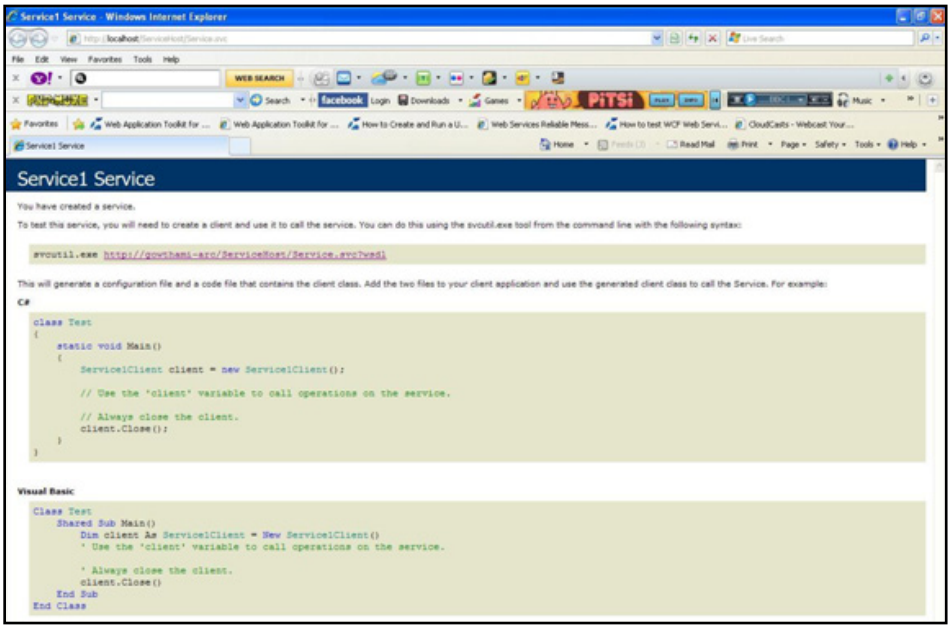


Figure 37: Launching the Service

- Now, add the service address in the **Add Service Reference** window.
- Once the address is given, it fetches all the available services on the server (Figure 38).

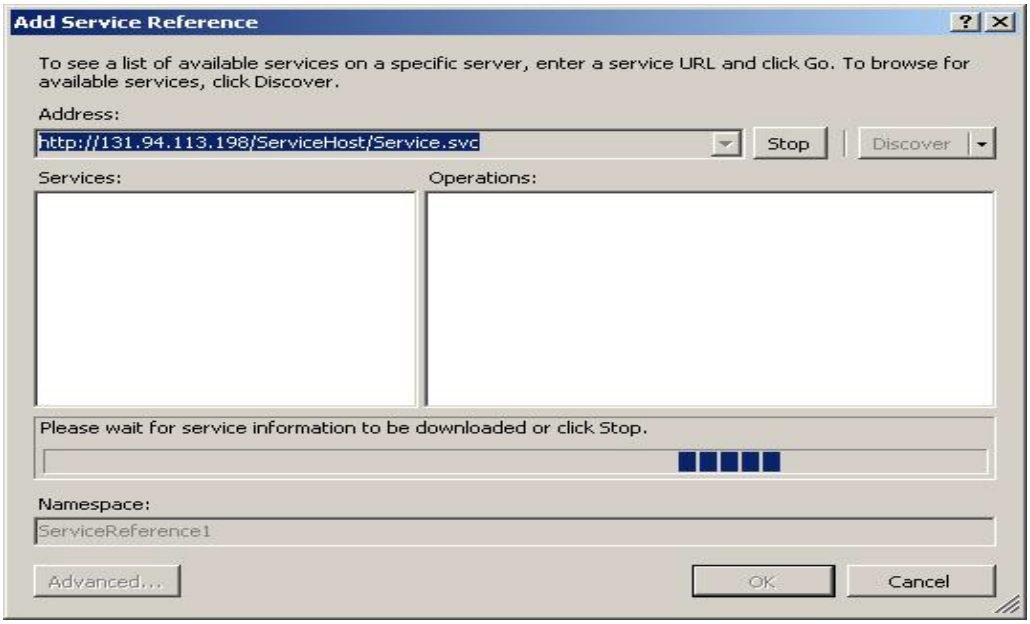


Figure 38: Providing the Service Address

- Select the service which is appropriate from the provided list of services (Figure 39).

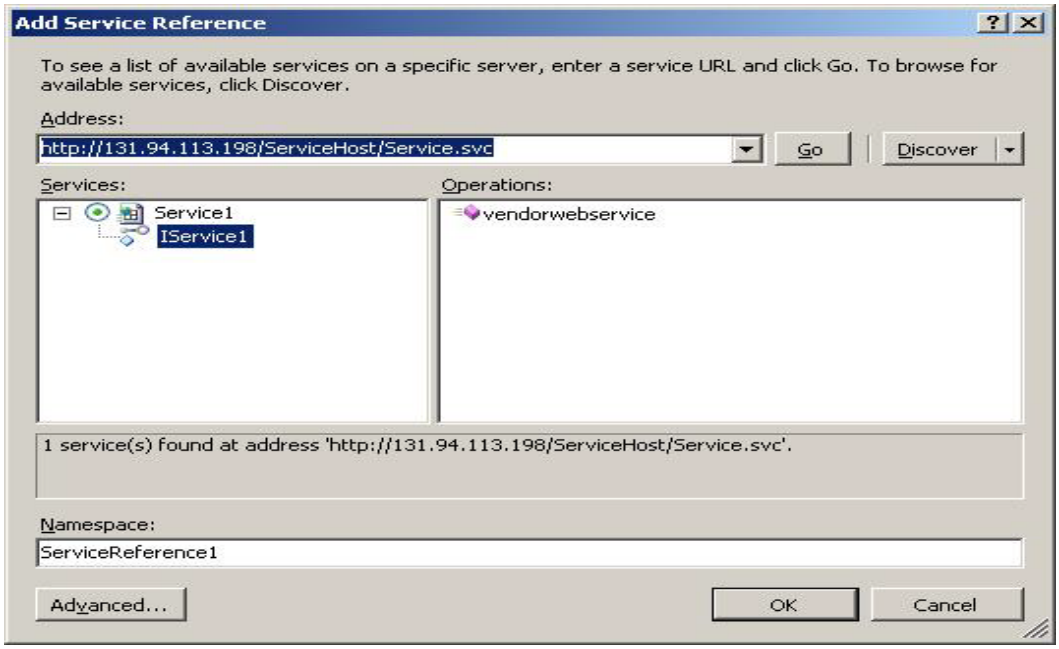


Figure 39: Selecting the Appropriate Service

- It downloads the metadata and generates the code that can be used in the client application. It also brings namespaces needed for the WCF service library, like System.ServiceModel.
- Finally, design the client's web page; create a proxy for the service to get the information from the service which can be displayed in the client, as the one shown in Figure 40.

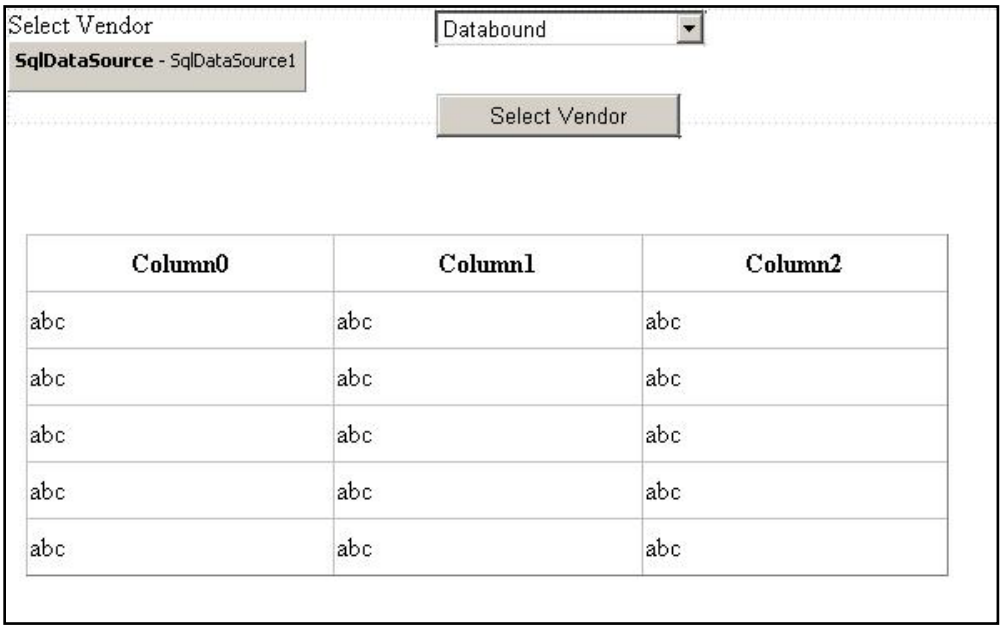


Figure 40: Web Client Interface

5.1.2 *Creating Windows Form Client*

- Right click in the solution explorer.
- Expand **Add.**
- Select **New Project.**
- This pops up the **Add New Project** window.
- Expand the **Visual C#** in the **Windows templates.**
- Select **Windows Forms Application.**

- Name it **WindowsFormsClient.**
- The location of the project **C:\DocumentsandSettings\harini\MyDocuments\Visualstudio2008\Projects\WcfClientLibrary2\WebClient** and
- Hit **OK** (Figure 41).

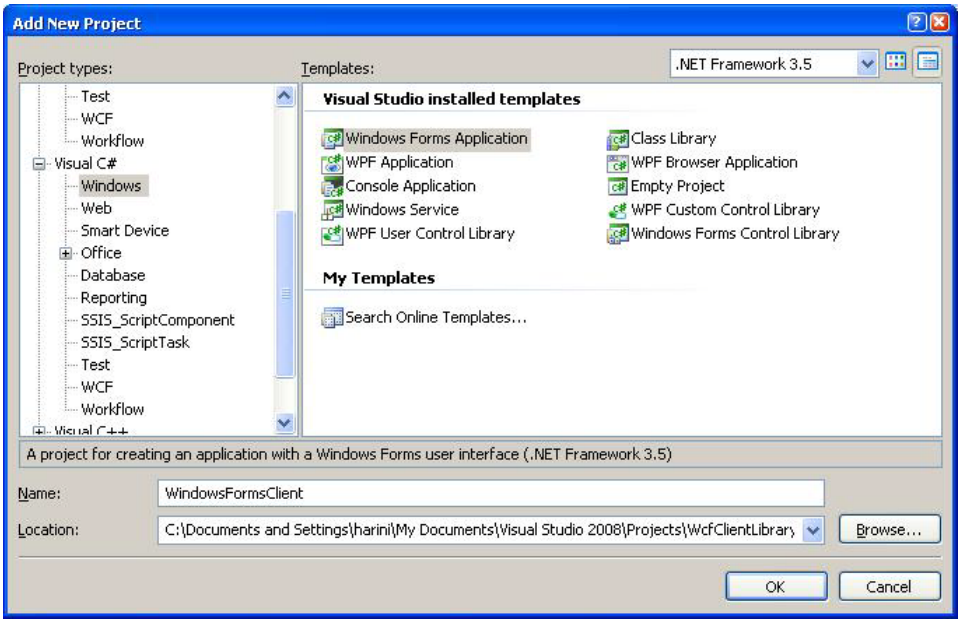


Figure 41: Creating a Windows Forms Client Application

- The project template provides a designer and code behind files (Figure 42).

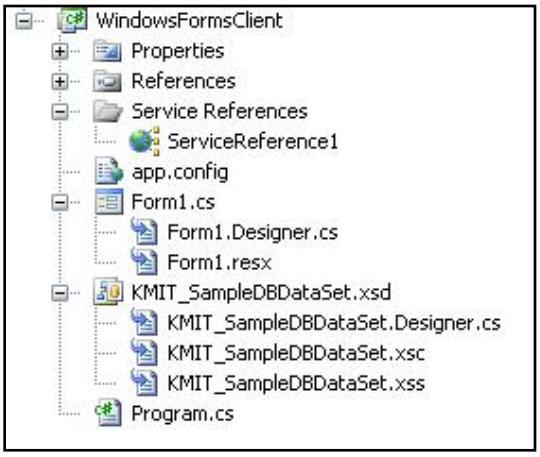


Figure 42: Windows Forms Client Project Template

- Next, generate a service reference that knows how to communicate with the service by right clicking on the references in the **WindowsFormsClient** project (Figure 43).

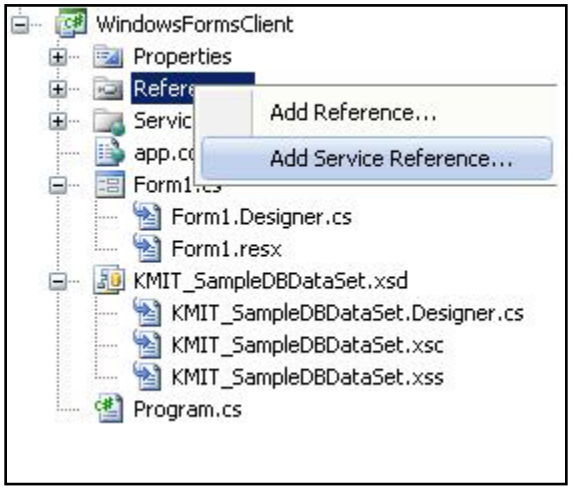


Figure 43: Adding Service Reference

- Before adding the service reference, launch the service library.
- Once the service is launched, copy the address to the clipboard (Figure 44).

Figure 44: Launching the Service

- Now, add the service address in the **Add Service Reference** window.

- Once the address is given, it fetches all the available services on the server (Figure 45).

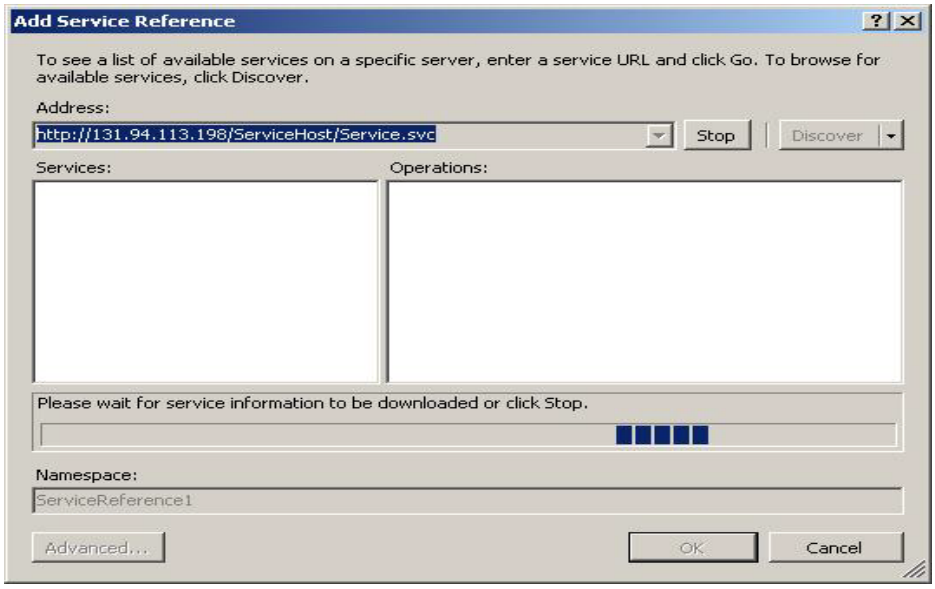


Figure 45: Providing the Service Address

- Select the service which is appropriate from all the provided list of services (Figure 46).

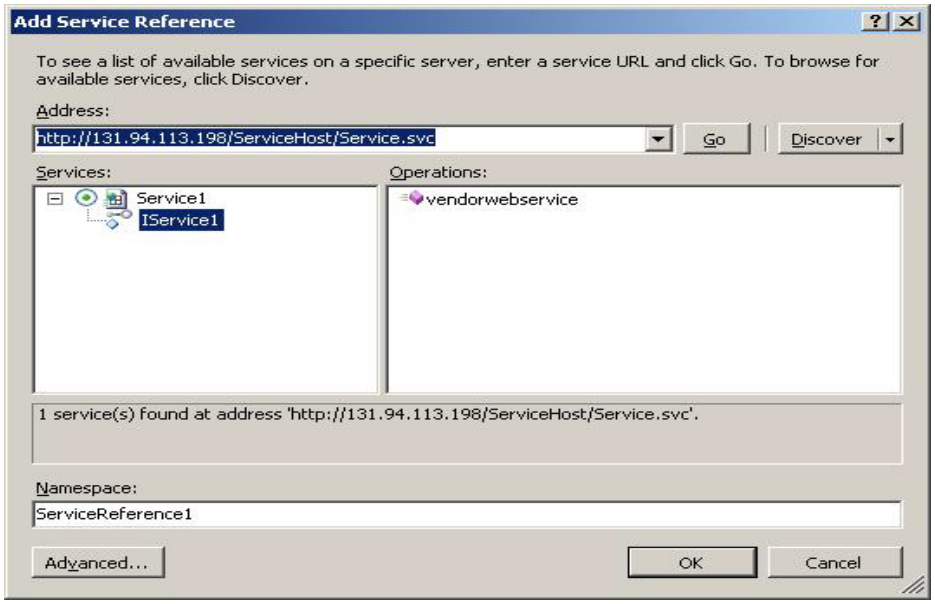


Figure 46: Selecting the Appropriate Service

- It downloads the metadata and generates a code that can be used in the client application. It also brings the namespaces needed for the WCF service library, like System.ServiceModel.
- Finally, design the client's forms application page and pull the information from the service to be displayed, as shown in Figure 47.

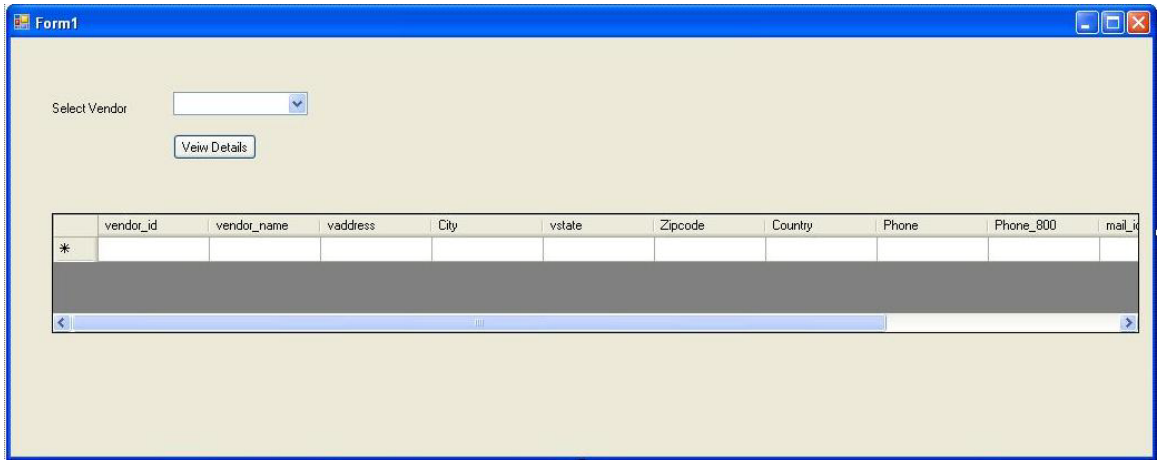


Figure 47: Designing Windows Forms Client Interface

5.1.3 *Creating Mobile Web Application*

- Right click in the solution explorer.
- Expand **Add.**
- Select **New Project**.
- This pops up the **Add New Web Site** window.
- Expand the **Visual C#** in the **Web templates.**
- Select **ASP.NET Web Site**.
- Name it as **MobileWeb.**

- The location of the project as **C:\DocumentsandSettings\harini\MyDocuments\Visualstudio2005\WebSites\MobileWeb** and
- Hit **OK** (Figure 48).

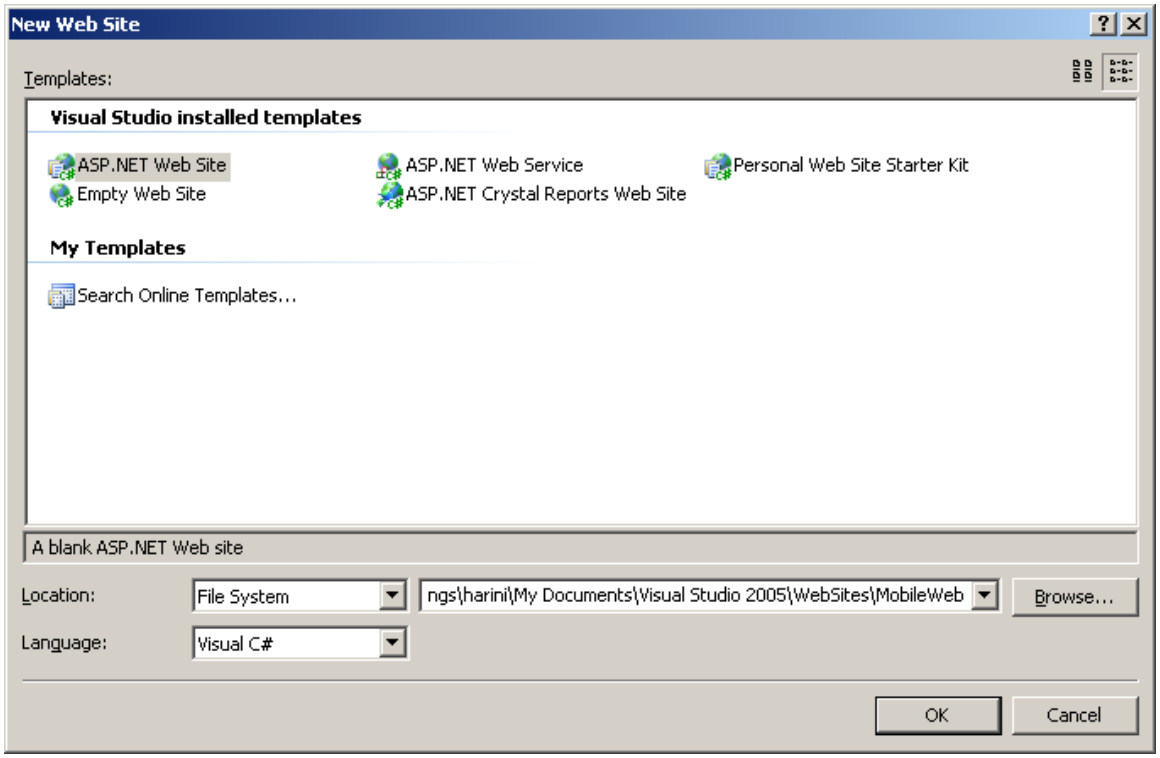


Figure 48: Creating a Mobile Web Client Application

- The project template provides a designer and code behind files (Figure 49).

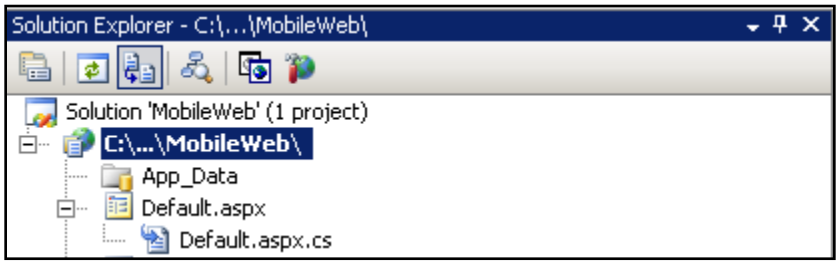


Figure 49: Mobile Web Client Project Template

- Right click on the web site

- New item is added to **MobileWeb** project (Figure 50).

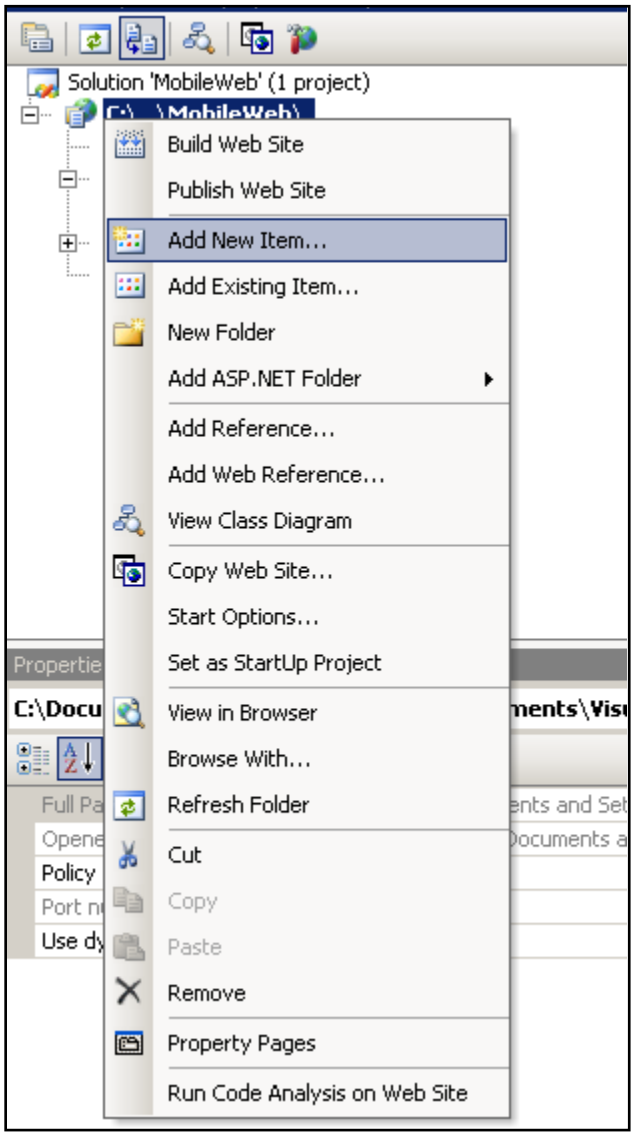


Figure 50: Adding New Item

- This opens up all the templates for adding a new item.
- Mobile web form is selected (Figure 51).

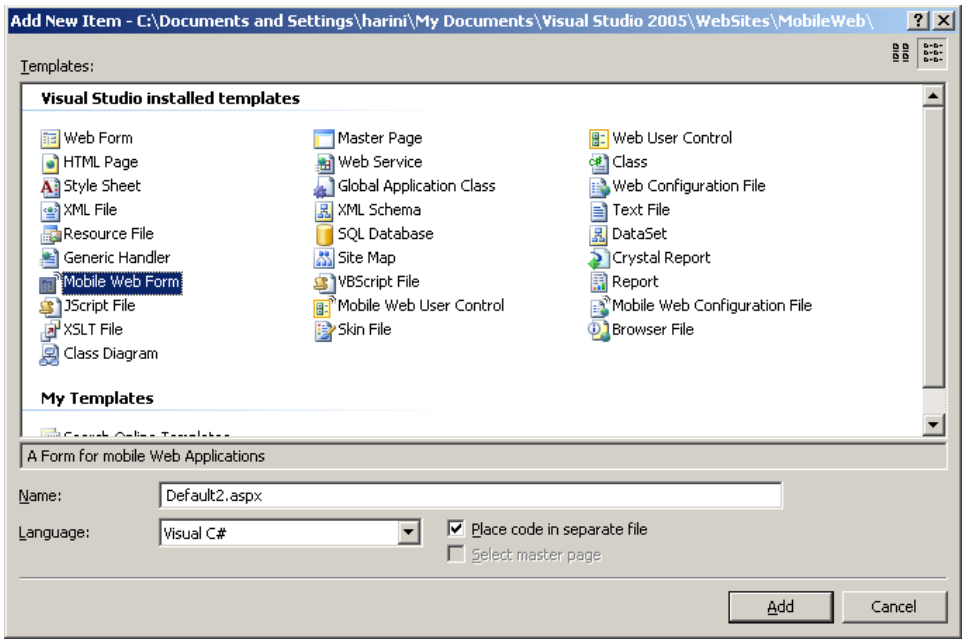


Figure 51: Adding Mobile Web Form

- A mobile web form is added to the project. Figure 52 shows the project template.

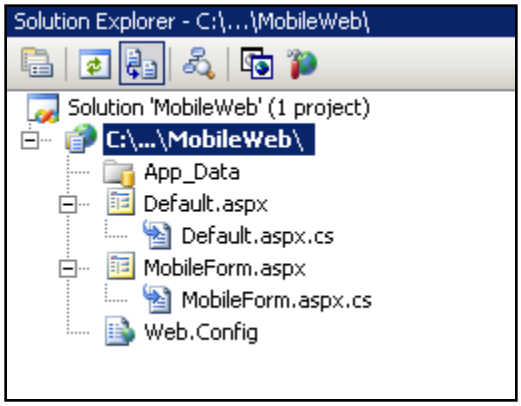


Figure 52: Project Template

- Finally, design the client's mobile web form and create a proxy for the service to get the information from the service, which can be displayed in the client, as shown in Figure 53.

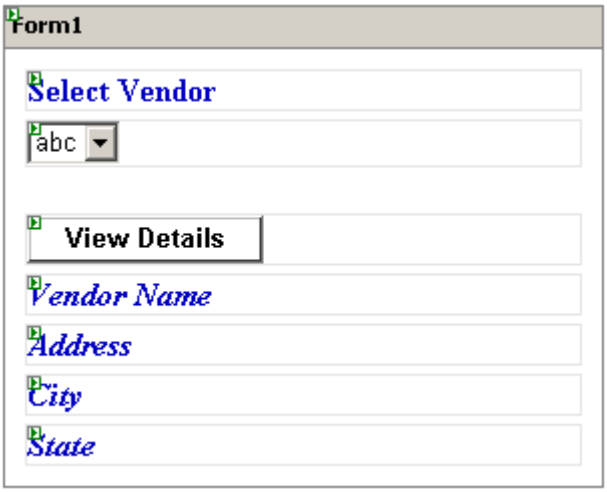


Figure 53: Mobile Web Interface

5.2 Testing

Two types of tests are conducted on the VM Web Service application. They are:

- Unit Test
- Load Test

5.2.1 *Unit Test*

Unit testing improves the application manageability by finding the bugs as soon as they are introduced into the code. Generally, testing applications written in ASP.NET are performed using visual studio team systems, but visual studio 2008 provides with an ability to write the unit tests for the application. For performing the unit test on the VM Web Service application, visual studio 2008 is used.

- First of all, a test project is added by right clicking on the solution explorer.
- Select **Add.**
- Click on **New Project** (Figure 54).

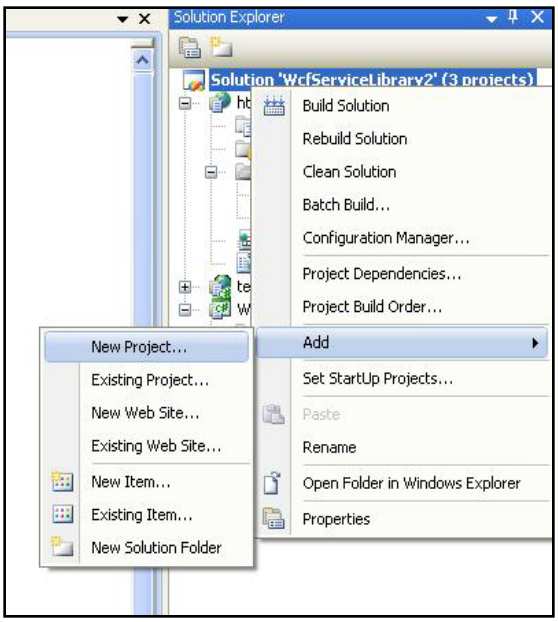


Figure 54: Adding a Test Project

- Select **Test Project** template from the **Add New Project** dialog box.
- Give the name as Test Project and the location of the project as **C:\Documents andSettings\gthot001\MyDocuments\VisualStudio2008\Projects\WcfServiceLibrary2\TestProject\TestProject.csproj,** as shown in Figure 55.

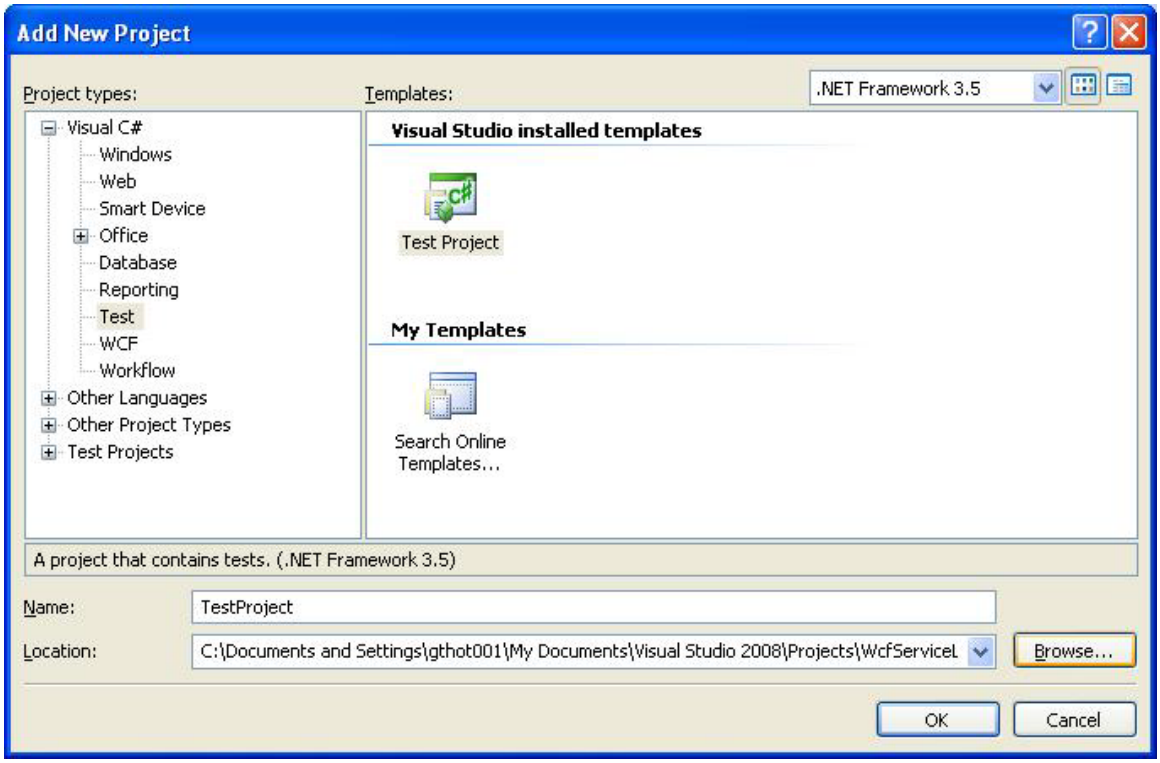


Figure 55: Naming and Locating the Project

- Once the test project is created, add a new unit test project to the test project by right clicking on the **Test Project** (Figure 56).

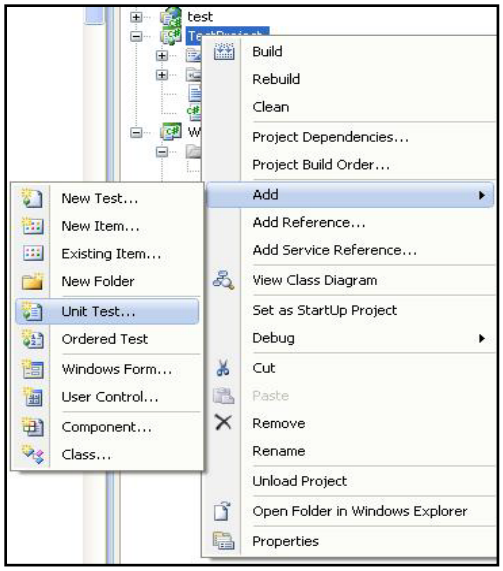


Figure 56: Adding a Unit Test Project

- Once the Unit test project is created, it takes to the “Create Unit Tests” window. The appropriate methods are selected, for which unit test case should be generated (Figure 57).

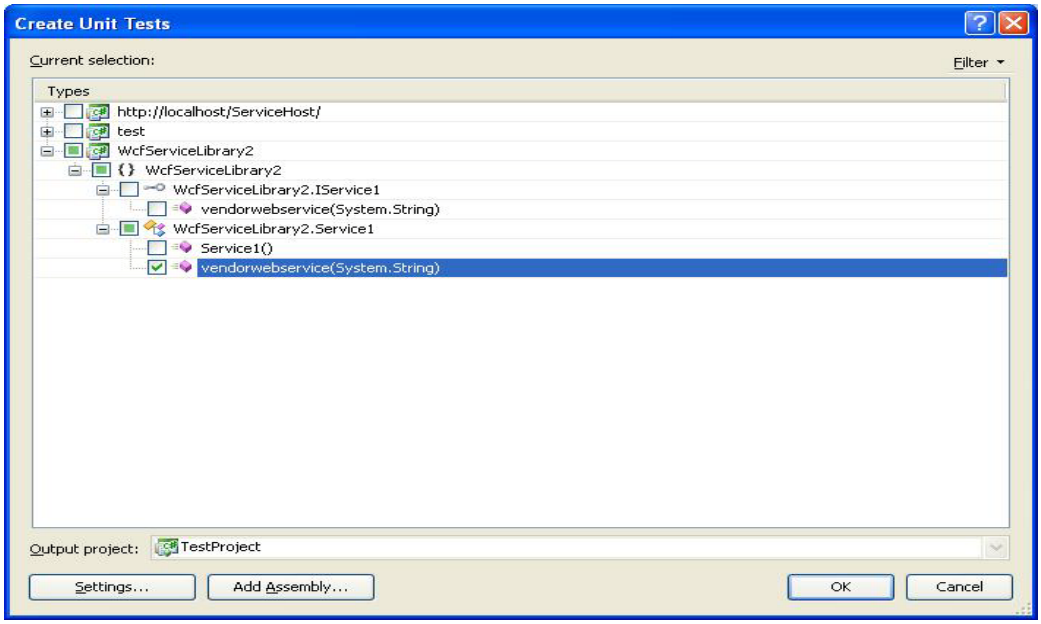


Figure 57: Class Selection

- Once the selection is made, unit test class is generated for the selected methods (Figure 58).

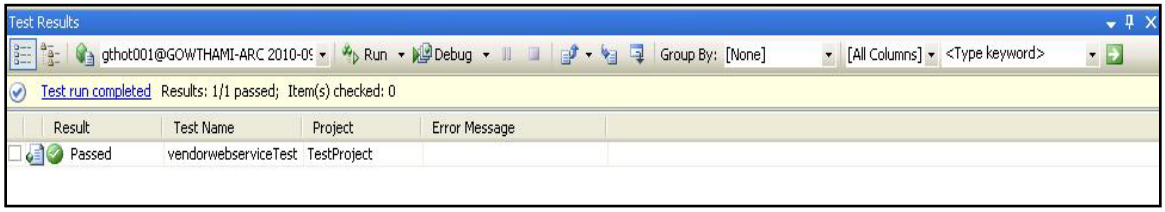
```
/// <summary>
///A test for Page_Load
///</summary>
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Documents and Settings\\gthot001\\My Documents\\Visual Studio 2008\\
[UrlToTest("http://localhost:49566/")]
[DeploymentItem("VendorClient.dll")]
public void Page_LoadTest()
{
    _Default_Accessor target = new _Default_Accessor(); // TODO: Initialize to an appropriate value
    object sender = null; // TODO: Initialize to an appropriate value
    EventArgs e = null; // TODO: Initialize to an appropriate value
    //target.Page_Load(sender, e);
   // Assert.Inconclusive("A method that does not return a value cannot be verified.");
}

/// <summary>
///A test for ImageButton3_Click
///</summary>
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Documents and Settings\\gthot001\\My Documents\\Visual Studio 2008\\
[UrlToTest("http://localhost:49566/")]
[DeploymentItem("VendorClient.dll")]
public void ImageButton3_ClickTest()
{
    _Default_Accessor target = new _Default_Accessor(); // TODO: Initialize to an appropriate value
    object sender = null; // TODO: Initialize to an appropriate value
    ImageClickEventArgs e = null; // TODO: Initialize to an appropriate value
    target.ImageButton3_Click(sender, e);
   // Assert.Inconclusive("A method that does not return a value cannot be verified.");
}

/// <summary>
///A test for GridView1_SelectedIndexChanged
///</summary>
[TestMethod()]
[HostType("ASP.NET")]
[AspNetDevelopmentServerHost("C:\\Documents and Settings\\gthot001\\My Documents\\Visual Studio 2008\\
[UrlToTest("http://localhost:49566/")]
[DeploymentItem("VendorClient.dll")]
public void GridView1_SelectedIndexChangedTest()
```

Figure 58: Test Class

Results Summary



5.2.2 *Load Test*

Load testing an application ensures that it will function correctly once it's in production. Performance problems are common, and have a wide variety of causes like software configuration issues, poor network configuration, software code and insufficient

hardware resources. The only way to figure out these problems before the application goes into production is by simulating a large number of simultaneous users. Load testing is done using Neoload. The professional load testing software provides all the features needed to carry out load tests and analyze the results. Neoload is able to record business actions performed within a web application, such as submitting a form or carrying out a search. These actions can be played back by as many virtual users as required to simulate the load that the server will have to bear.

Neoload has two main components: the controller and the load generator. A load generator is included with the controller to make deployment even simpler. During a test, Neoload collects information on the server infrastructure via its monitoring modules to pinpoint the causes of performance issues. The controller provides a graphical interface from which the user can create record scenarios, run tests and analyze the results. The user executes the test by controlling the load generators outputs. The load generator simulates users accessing the application being tested, collects information on the applications performance and deploys at no additional cost on as many machines as required [28]. The results after load testing the VM Web Service are as below.

Results summary:

Project	VendorManagement_Performance	Load Policy: Population is constant with 10 users
Scenario	Scenario1	
Start date	Sep 2, 2010 6:57:26 PM	Description: Vendor Service Performance
End date	Sep 2, 2010 6:59:26 PM	
Duration	00:02:00	Filters: None
LG Hosts	Localhost	Debug: Disabled

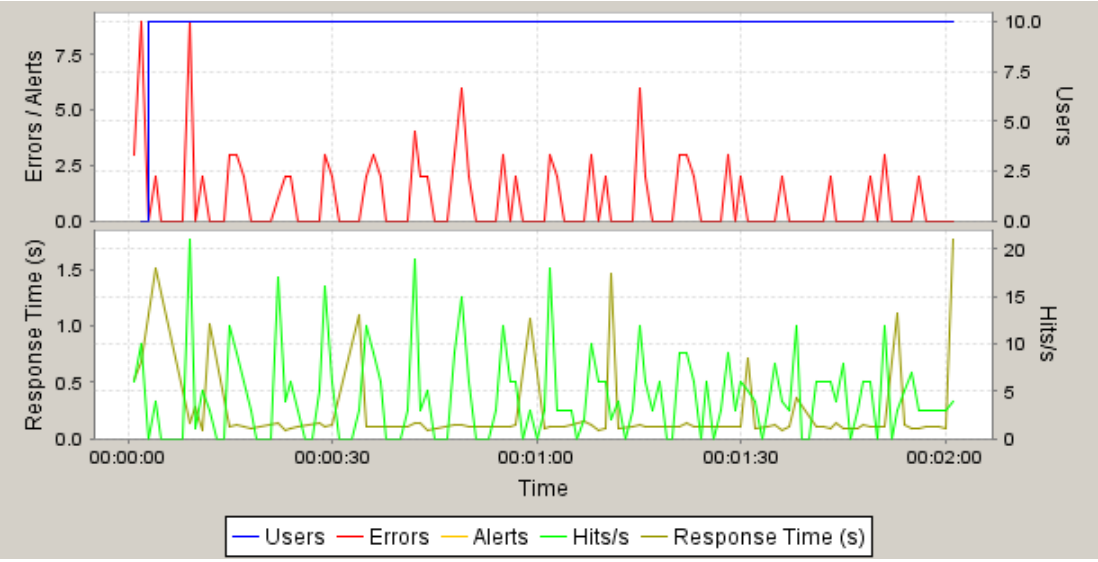
The above table gives the results summary of the load test conducted. It captures the date, starting time, ending time, the duration of the test and the load policy of the test.

Statistics summary

Total pages: 168	Total duration alerts: 0%
Total hits: 528	Average pages/s: 1.4
Total users launched: 178	Average hits/s: 4.4
Total throughput: 0.73 MB	Average Request response time: 0.189s
Total hit errors: 116	Average Page response time: 0.394s
Total action errors: 0	Average throughput: 0.05Mb/s

The above table gives the statistics summary of the load test conducted. It records the total users launched, number of hits, hit errors, average hits per second, average

request response time, average page response time and average throughput.



General Statistics

Min	Avg	Max	Hits	Err	Med	Avg 90%	Std Dev
All virtual users							
0.28	0.55	4.33	168	69	0.334	0.415	0.669
All pages							
0.265	0.394	3.28	168	0	0.273	0.284	0.517
All requests							
<0.01	0.189	3.28	528	116	0.114	0.138	0.265

Populations

Virtual User	Select Vendor
Percentage	100%

Upload	Unlimited
Download	Unlimited
Browser	Recorded one
Handle cookies	Yes
Connections	2
Handle Cache	As recorded

Virtual Users

Min	Avg	Max	Hits	Err	Med	Avg 90%	Std Dev
Select Vendor							
0.28	0.55	4.33	168	69	0.334	0.415	0.669

Select Vendor

Min	Avg	Max	Hits	Err	Med	Avg 90%	Std Dev
Select Vendor							
0.28	0.55	4.33	168	69	0.334	0.415	0.669
/api/v1.0/pnr							
Min	Avg	Max	Hits	Err	Med	Avg	Std Dev
0.265	0.394	3.28	168	0	0.273	0.284	0.517

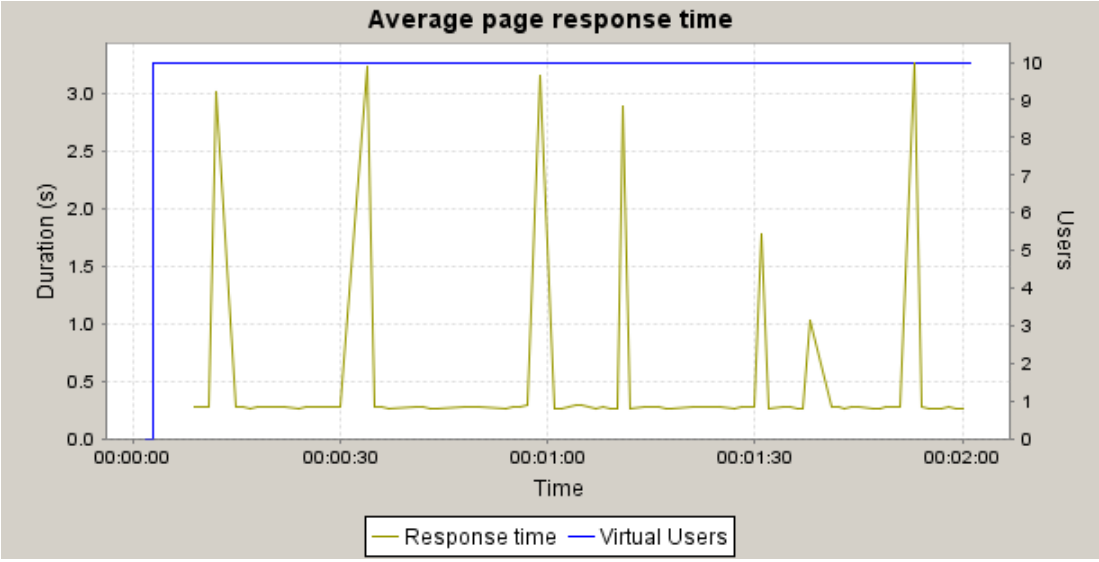
Neoload Performance counters

Min	Avg	Max	Med	Avg 90%	Std Dev
Neoload/User Load					
10	10	10	10	10	0

LG localhost Performance counters

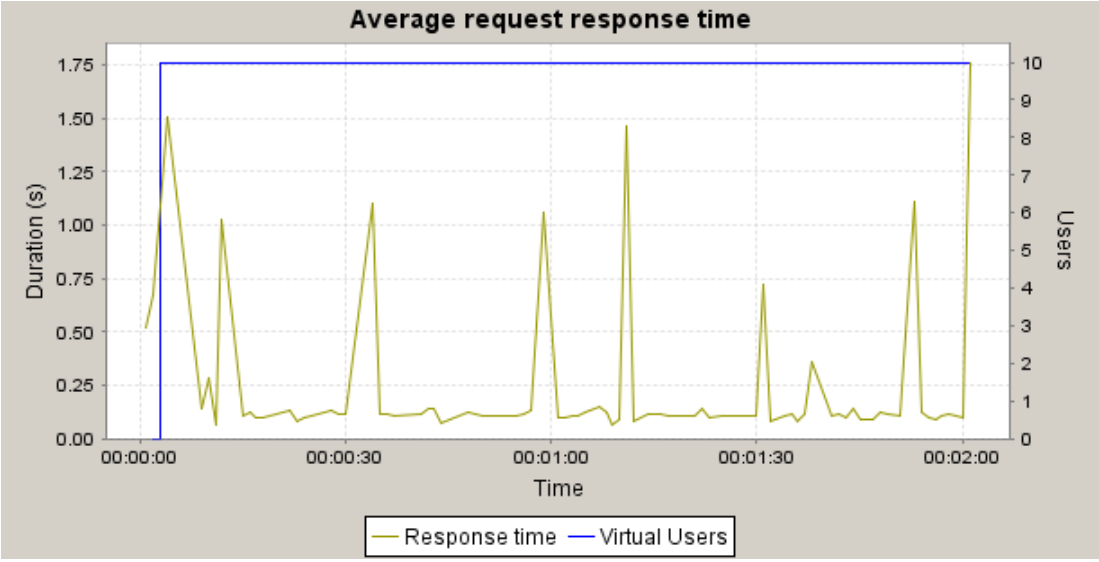
Min	Avg	Max	Med	Avg 90%	Std Dev
LG localhost/CPU					
2	7.32	46	6	6.73	5.19
LG localhost/Memory					
4	10.03	15	10	10.07	3.06
LG localhost/Throughput					
0	0.049	0.216	0.035	0.045	0.047
LG localhost/UserLoad					
10	10	10	10	10	0
LG localhost/Population1/UserLoad					
10	10	10	10	10	0
LG localhost/Population1/SelectVendor/UserLoad					
10	10	10	10	10	0

Main Graphs



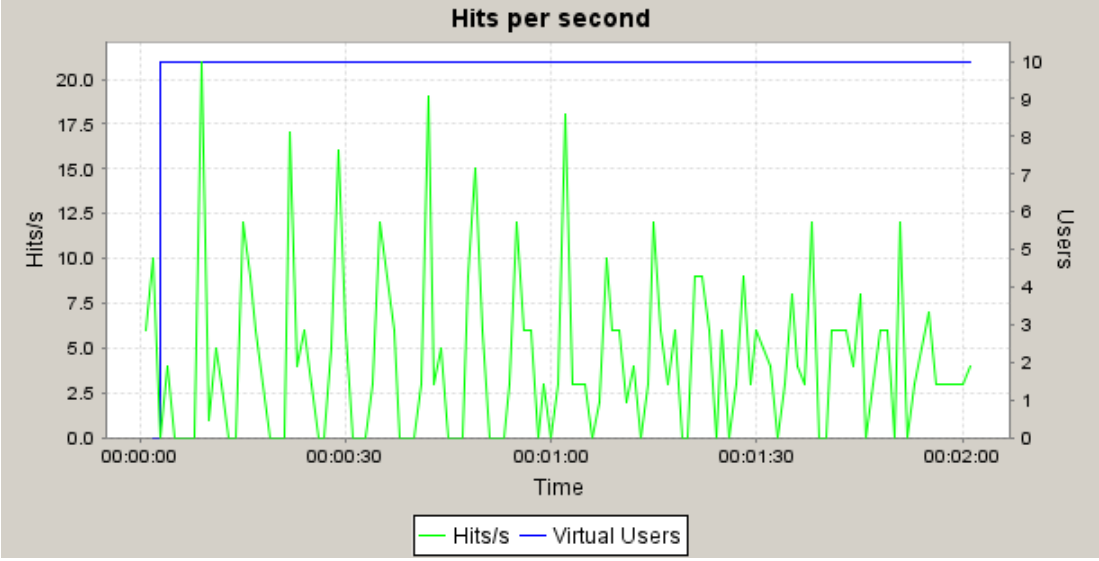
The above graph displays the average response time, in seconds, of all pages during the test.

Graph Min	Average	Graph Max	Graph Median	Graph Avg 90%	Graph Std.Dev.
0.265	0.394	3.265	0.273	0.285	0.517



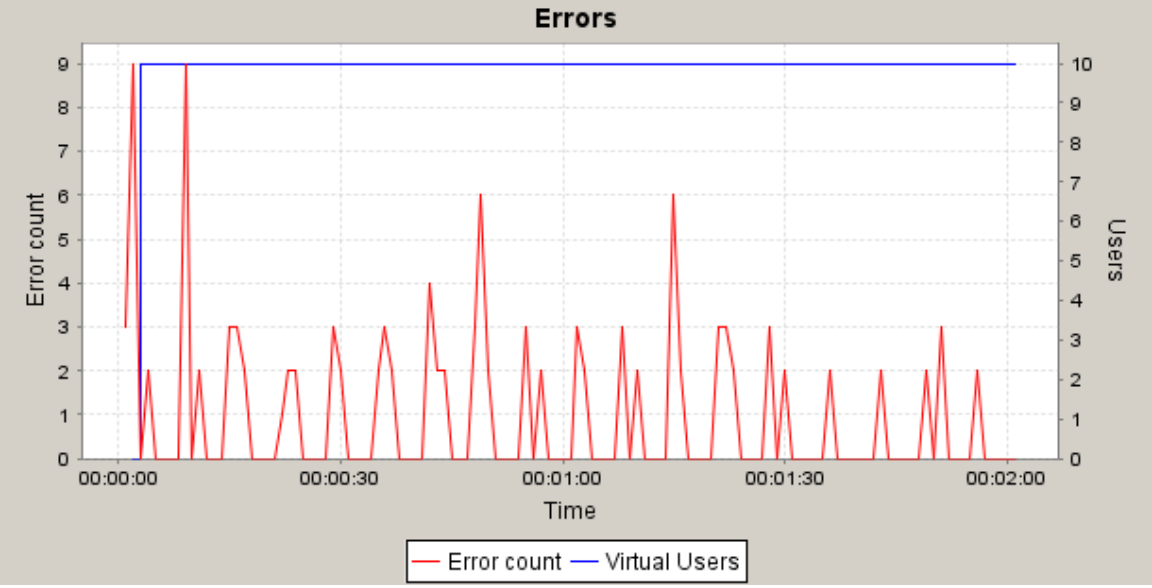
The above graph displays the average response time, in seconds, of all pages during the test.

Graph Min	Average	Graph Max	Graph Median	Graph Avg 90%	Graph Std.Dev.
0.068	0.189	1.762	0.114	0.138	0.265



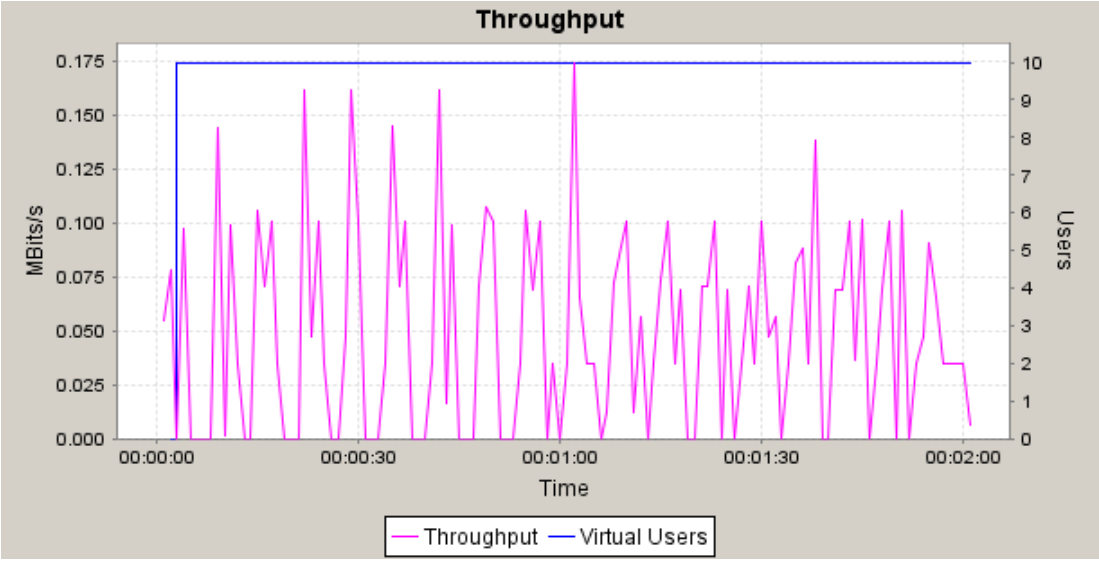
The above graph displays the number of hits on the server by virtual users.

Graph Min	Average	Graph Max	Graph Median	Graph Avg 90%	Graph Std.Dev.
0	4.4	21	3	3.8	4.6



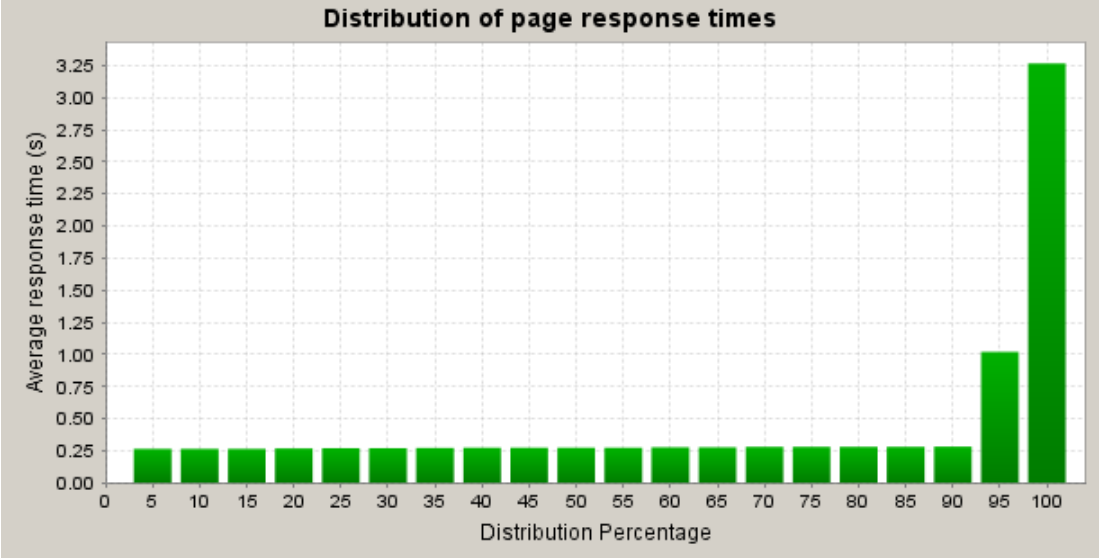
The above graph displays the error rate in errors per sampling interval.

Graph Min	Average	Graph Max	Graph Median	Graph Avg 90%	Graph Std.Dev.
0	1	9	0	0.7	1.7



The above graph displays the number of megabits of data per second returned by the server.

Graph Min	Average	Graph Max	Graph Median	Graph Avg 90%	Graph Std.Dev.
0	0.05	0.17	0.03	0.04	0.05



The graph above displays the percentage of pages that were performed within a given time range. This graph helps determine the percentage of pages that meet a performance objective. For example, it might show that 90% of the pages have a response time under n seconds.

5.3 Experiments

This section describes all the experiments conducted. So far, the service has been built, reliability features are implemented and a client is developed, which can access the service. The next step is to conduct some experiments to make sure that the reliability

features are enabled in the service. To conduct this experiment, we call the service from the client application and run the FIDDLER in the background. The FIDDLER intercepts and captures all the traffic from the client to the service and from the service to the client. Once the fiddling is completed, we can observe the statistics and the inspectors which contain the XML format for the captured data, and see if the service is correctly enabled with reliable sessions. The next experiment is a fault-case scenario where we test the service for the tolerance of failures. To experiment, in this case, we send a request from the client to the service and once the request reaches the service, we stop the service for a certain interval. In real time scenarios this interval may be very long; after that interval of time, we start the service again. Once the service starts, it handles the requests, which were in queue, i.e. it handles the request that were sent before the service went down. This ensures the reliability of the Vendor Management Web Service. This makes sure that the client gets the requested data, even when the service is down for a certain period of time.

5.3.1 *Experimental Result-1*

This is a scenario to test whether the reliable sessions are enabled in the VM Web Service or not.

Output:

The user selects the **Vendor_name** and hits **View Details** button (Figure 59). The service responds back sending the information of the selected vendor, which includes Vendor id, Vendor name, address, city, state, zip code, country, phone, mailID, website and comments (Figure 60).

Figure 59: Client Sending Request to Service

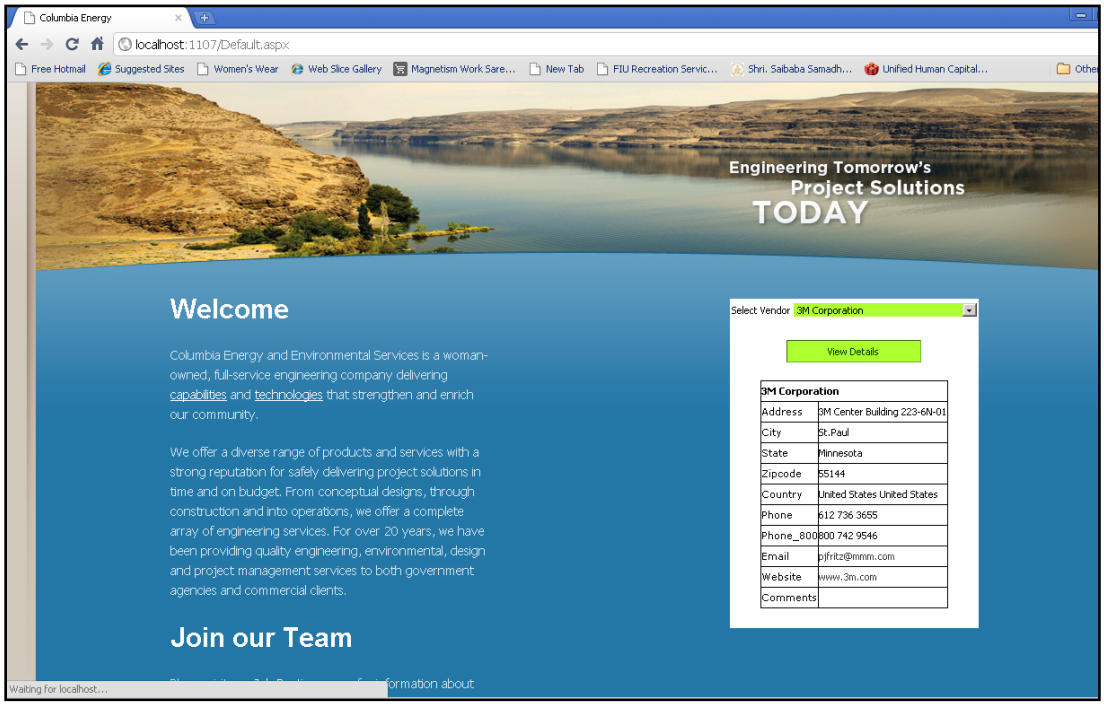


Figure 60: Client Receiving Response from the Service

Background process/ Discussion:

The client application sends a request to the service. Once the service receives the request, a service response is sent from the service to the client. In the background, FIDDLER is run to capture all the traffic. With a button click, the client requests the creation of a sequence by sending a **CreateSequence** message. The service responds with **CreateSequenceResponse** message, which assigns a unique sequence **Identifier**.

The WS-RM specification supports **CreateSequence / CreateSequenceResponse** request-response message pair to allow a service to initiate either:

- A one-way output sequence
- A duplex sequence, which is a pair of one-way output and one-way input sequences.

Each message that requires reliable delivery includes a sequence header block. The sequence header block contains a unique **Identifier** for the sequence, and each message in the sequence is assigned a unique **MessageNumber**. The service acknowledges successful receipt of messages by including the **SequenceAcknowledgement** header block.

Figure 61 shows the statistics captured by FIDDLER. It shows the handshake operations performed between the client and service. The client first gets connected and starts requesting the service. Once the service gets connected, it processes the request and sends the response to the client. Finally, client begins the response and completes the response once it gets the complete response from the service.

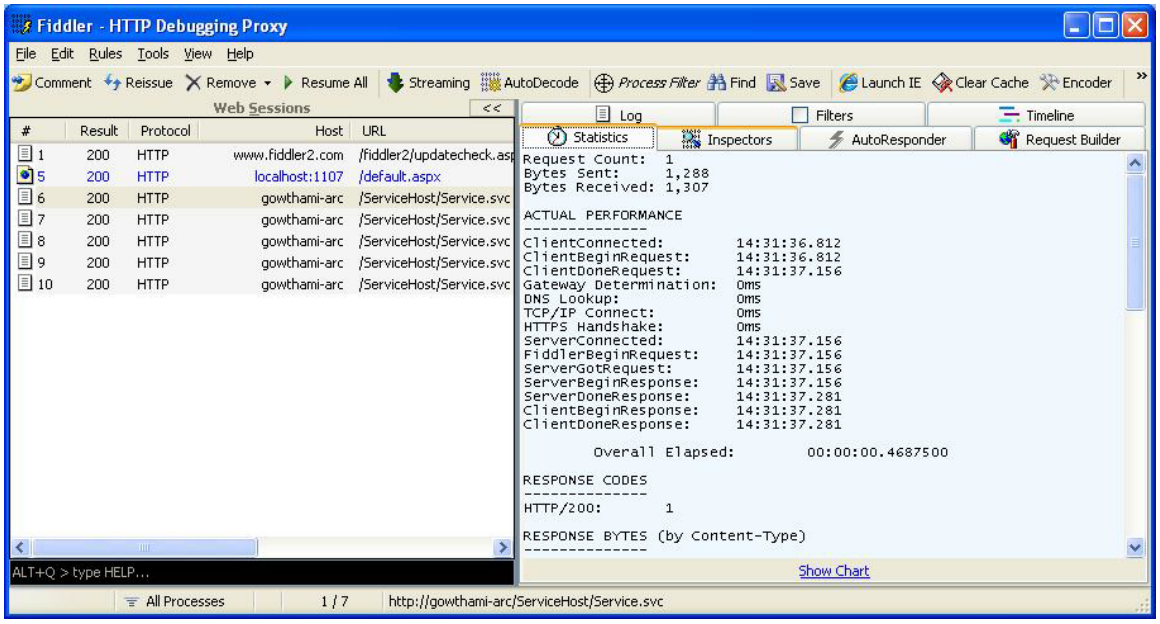


Figure 61: Statistics Captured By FIDDLER

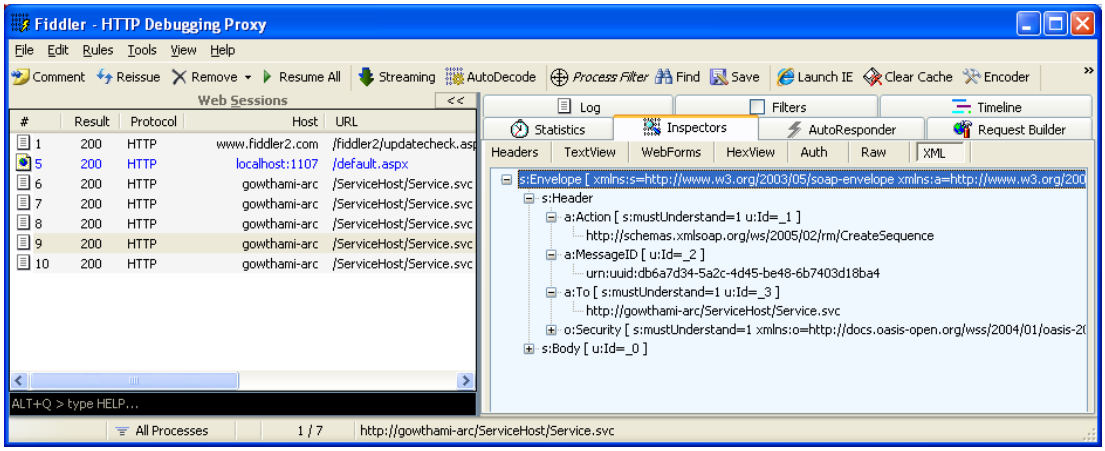


Figure 62: Client Request for Sequence Creation

All the WS-RM protocol elements are assigned with the namespace URI **http://schemas.xmlsoap.org/ws/2005/02/rm**. Figure 62 shows the client requesting creation of a sequence by sending a **CreateSequence** message to the service, whose address is **http://gowthami-arc/ServiceHost/Service.svc.**

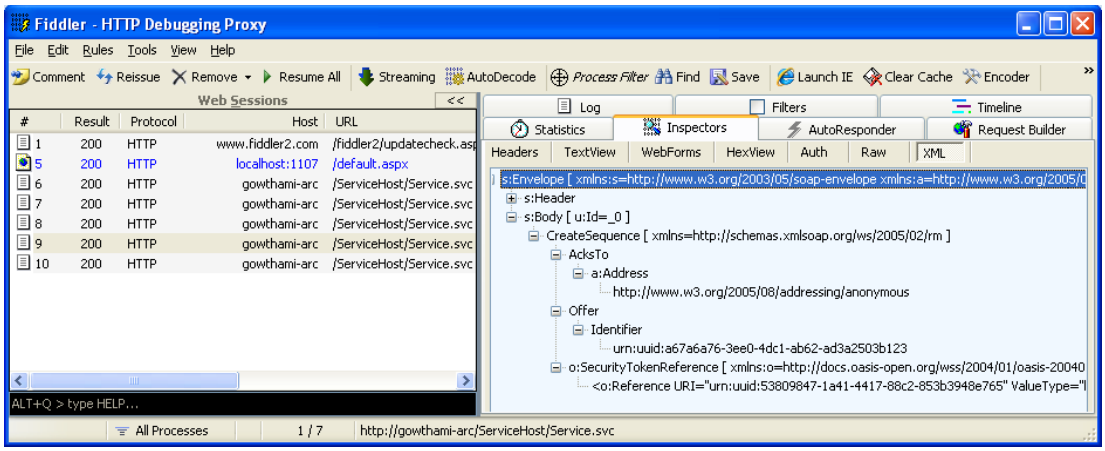


Figure 63: Service Acknowledgement to Client

The client is recognized with this endpoint address (**http://www.w3.org/2005/08/addressing/anonymous)**.Because some endpoints cannot be located with a meaningful URI (Figure 64), this URI is used to allow such endpoints to send and receive messages.

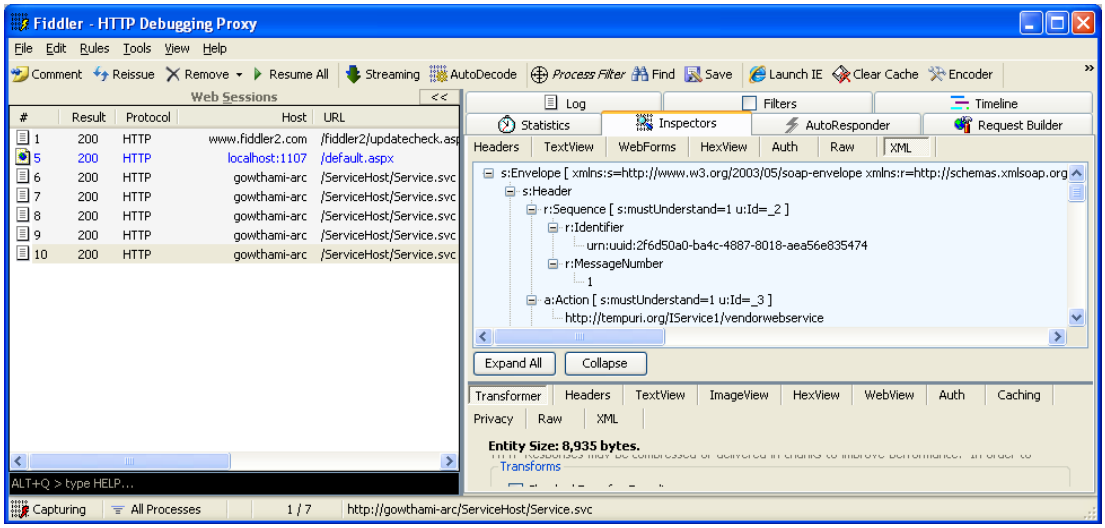


Figure 64: Sequence Header Block

To serve the request, service includes a sequence header block, as shown in Figure 64. This includes a unique **Identifier urn: uuid: 2f6d50a0-ba4c-8018-aea56e835474** for the sequence and each message in the sequence is assigned a unique **MessageNumber**: **1**. the message number increases monotonically by one for each

subsequent message in the sequence. This is the action done by the **vendorwebservice** operation located at **http://tempuri.org/IService1/vendorwebservice,** which is a **ReplyTo http://www.w3.org/2005/08/addressing/anonymous.**

Result: As we can see all the WS-RM specification terms from the results captured by the FIDDLER, we can confirm that the VM Web Service is enabled with reliable sessions of Windows Communication Foundation (WCF) and it adheres to the standards of WS-RM specifications.

5.3.2 *Experimental Result-2*

This is a scenario to check the tolerance of the service for failures. The client sends a request to a broken service and ultimately receives response once the service regains its state. This scenario is explained by comparing two situations. Once, when the service is not enabled with reliable sessions and the next situation is when the reliable sessions are enabled in the service.

Experimental Setup:

The client is a web application which is accessing the service. Both the client and the service are on two different machines and are located geographically apart from each other.

Output:

On selecting the vendor name and hitting the 'View Details' button, the client sends this request to the service (Figure 65), and the service responds back sending the information of the selected vendor, which includes Vendor id, Vendor name, address, city, state, zip code, country, phone, mailID, website and comments after it is up and running (Figure 66).

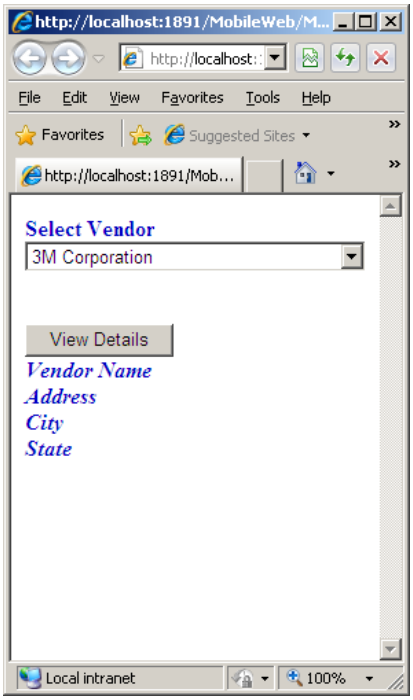


Figure 65: Mobile Client Requesting the Service

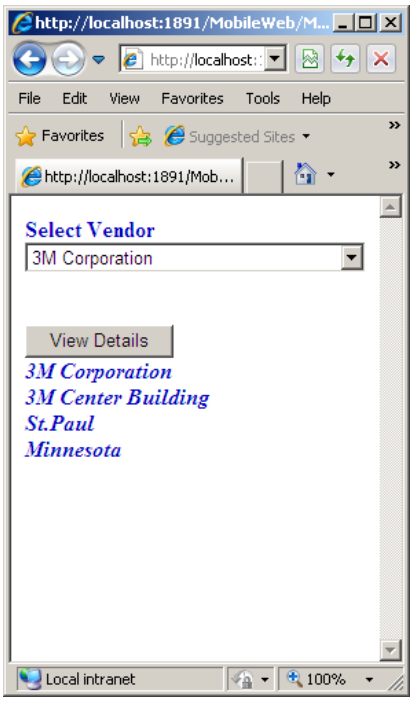


Figure 66: Service Response to the Client

Case 1:

On the service side, reliable sessions are disabled by changing the binding configuration. In binding configuration, the reliable session properties are set by changing the **ReliableSession Properties Enabled to False** as shown in the Figure 67.

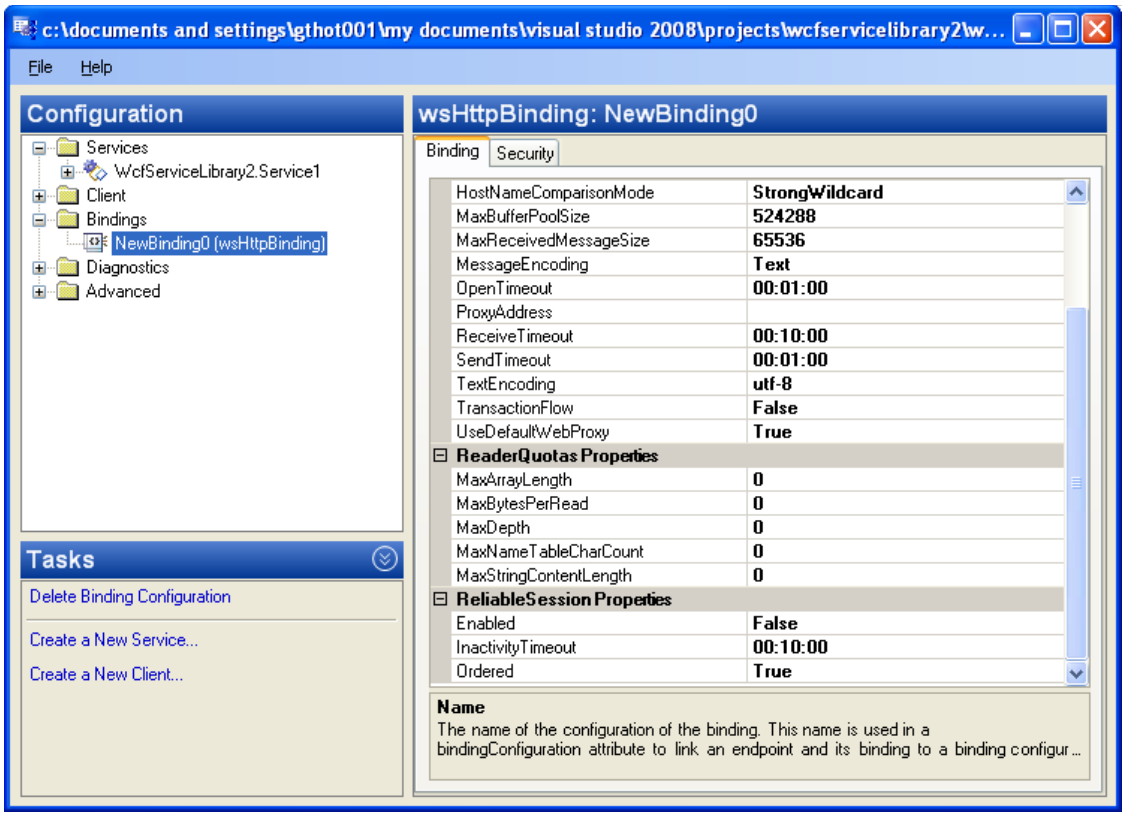


Figure 67: Disabling reliable sessions

Then client sends a request to the service. Once the request is sent, the service is stopped in Internet Information Services (Figure 68).To stop the service, **inetmgr** is run in the command prompt this opens up the Internet Information Services window. Expand **GOWTHAMI-ARC (local computer)** expand **Web Sites,** click on **Default Web Site** and then click on **Stop Item** in the top pane of the window as shown in Figure 68.

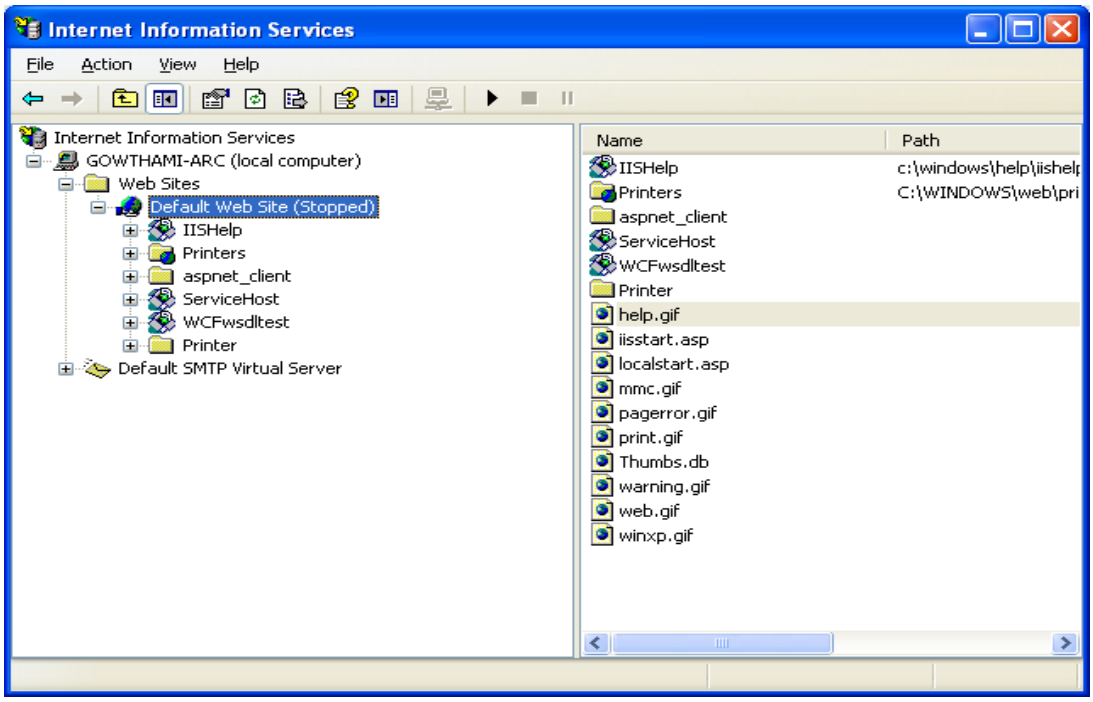


Figure 68: Stopping the Service

After a few seconds, the service is started again. To start the service, **inetmgr** is run in the command prompt. This opens IIS (Figure 69) window, expand **GOWTHAMI-ARC (local computer),** expand **Web Sites,** and click on **Default Web Site.** Then click on **Start Item** in the top pane of the window, as shown in Figure 69. Once the service is started, it throws an exception to the client as shown in the Figure 70.

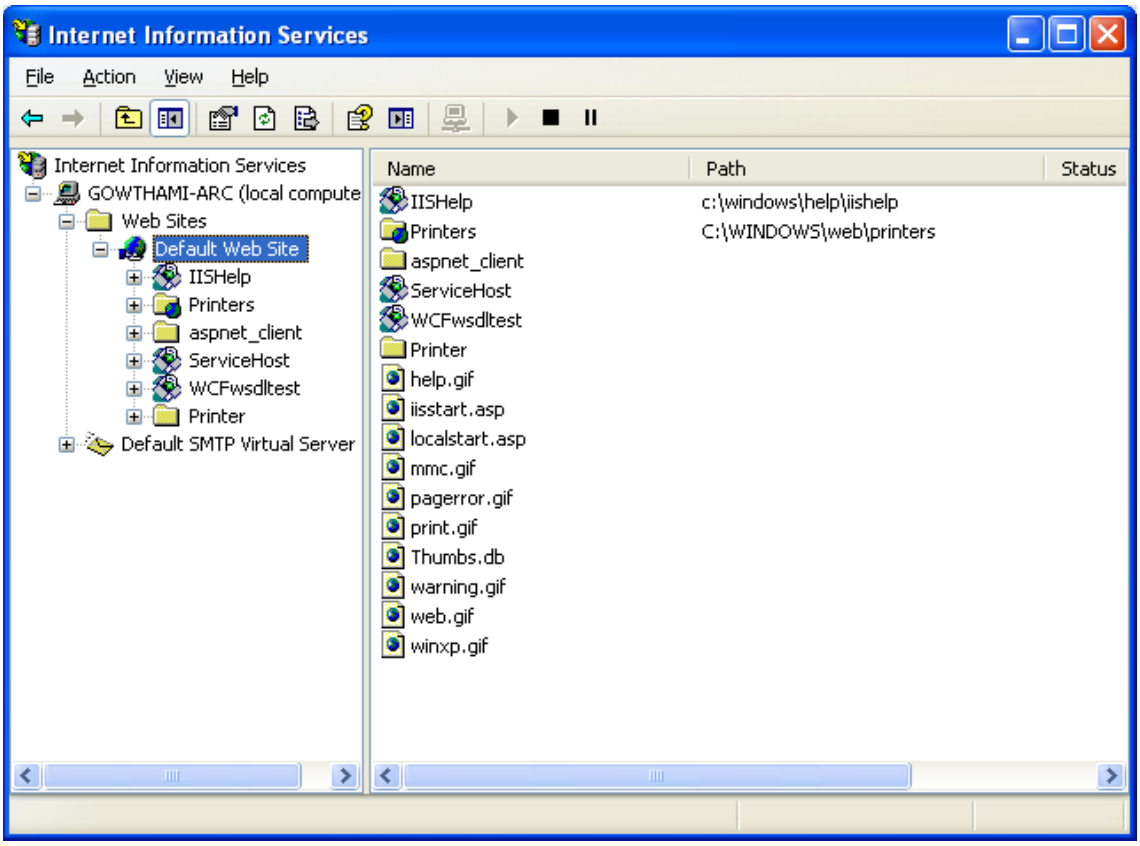


Figure 69: Starting the Service

Background process/ Discussion:

Since the service is not enabled with reliable sessions, it will not establish a cache (or queue) for storing the messages it received. There is no concept of queuing in this scenario. The requests which are made before the service is down are not saved in the queue. Messages are lost and there is no guarantee that the messages are processed. That is the reason why a protocol exception is generated.

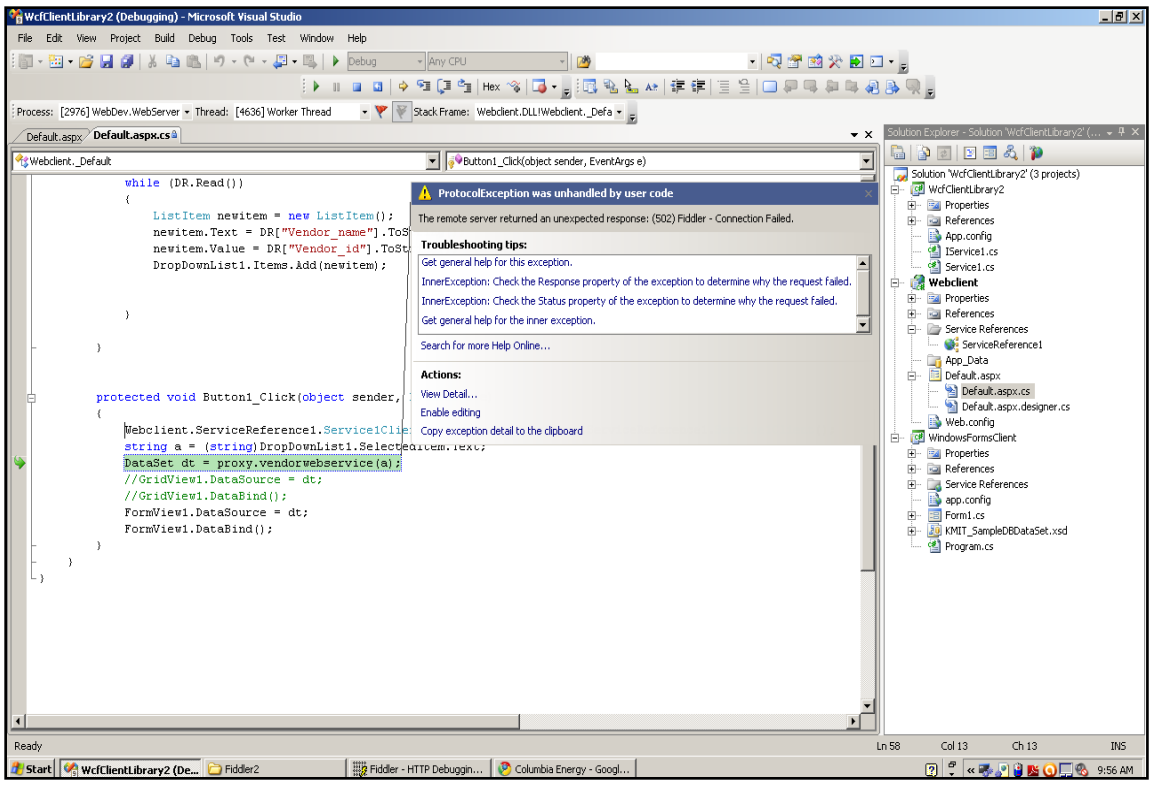


Figure 70: Exception

Case 2:

On the service side, reliable sessions are enabled by changing the binding configuration. In binding configuration, the reliable session properties are set by changing the **ReliableSession Properties Enabled to True** as shown in the Figure 71.

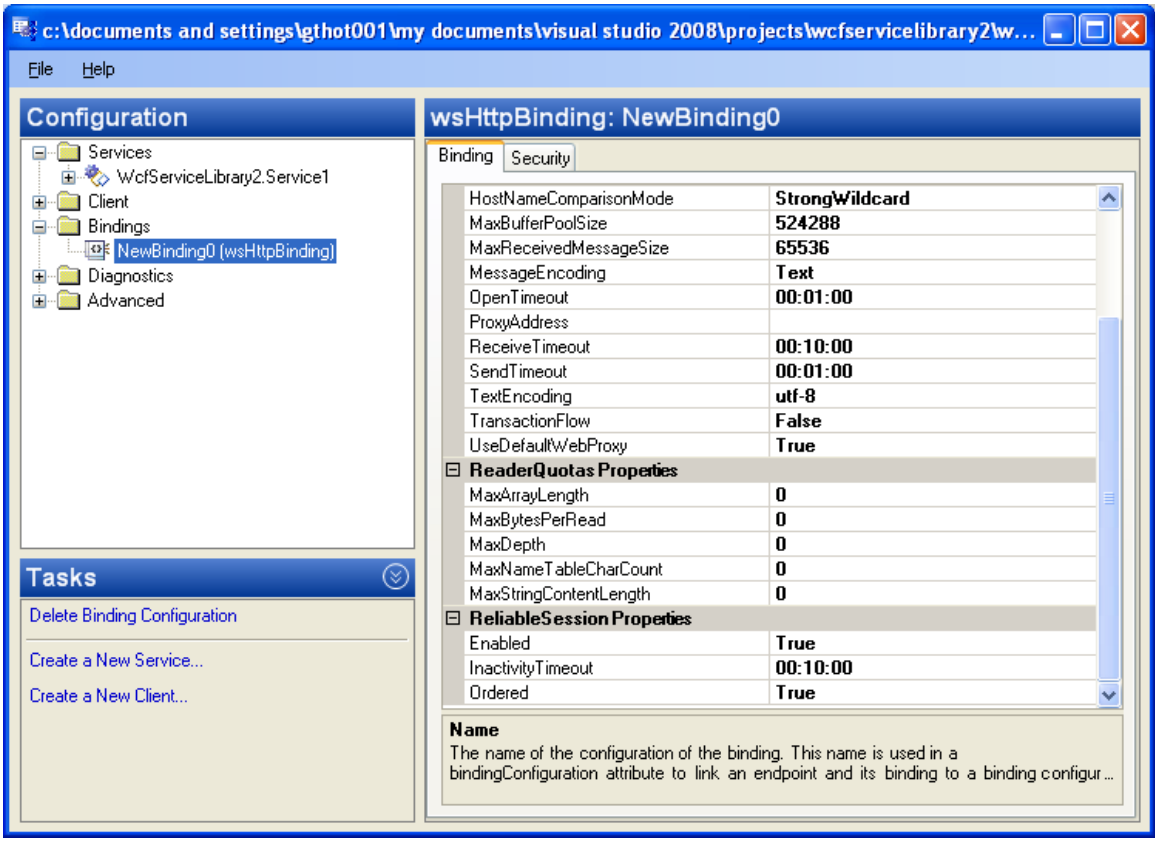


Figure 71: Enabling reliable sessions

Then client sends a request to the service. Once the request is sent, the service is stopped in Internet Information Services (Figure 68).To stop the service, **inetmgr** is run in the command prompt this opens up the Internet Information Services window. Expand **GOWTHAMI-ARC (local computer)** expand **Web Sites,** click on **Default Web Site** and then click on **Stop Item** in the top pane of the window as shown in Figure 68.

After a few seconds, the service is started again. To start the service, **inetmgr** is run in the command prompt. This opens IIS (Figure 69) window, expand **GOWTHAMI-ARC (local computer),** expand **Web Sites,** and click on **Default Web Site.** Then click on **Start Item** in the top pane of the window, as shown in Figure 69. Once the service is started, service returns the result to the client.

Background process/ Discussion:

Service establishes a cache for holding messages it accepted in order to store them before delivering them to the application. Deep down in WCF's channel architecture where the WS-RM channel is stacked on top of the transport channel, arriving messages are queued up in the transport channel as they arrive. So, even though when the service is down some times, it handles the request once when it is up and running. It processes the requests which are made before the service is down, and waiting in the queue to be processed. Whenever, a worker thread becomes available to process a message, the service model pulls a message from that transport queue through the channel stack (each channel pulls from its underlying channel) and dispatches the message into the VM Web Service. Then, the VM Web Service processes the request and sends the result back to the client.

By comparing these two situations, i.e. the service is enabled with the reliable sessions and the service is disabled with reliable sessions. There is guarantee that the messages are processed, when the service is enabled with reliable sessions and down for certain period of time. Reliable session make this possible by maintaining a queue of requests, which is not the case when the reliable sessions are disabled. There is no concept of queuing. Messages are lost and there is no guarantee that the messages are processed.

Result: From the above discussion, we can conclude that when the service is not enabled with reliable sessions there is no guarantee if the messages are processed and the result is sent back to client. But on the other side, when the service is enabled with reliable sessions we can ensure that the messages are processed and delivered to the client even

when the service is down. This describes the concept of reliability and guaranteed message delivery. By enabling reliable sessions in VM Web Service, we have successfully implemented reliability features in the service.

6. CONCLUSION

Implementing reliability features and the testing of reliability features are the two major challenges faced with the developed VM Web Service. In this scenario, the Vendor module is implemented as a Web Service for interoperability and integration. Creating applications that are later used as services, and communicating through messages have become commonplace and easy. Developers have to make sure that the messages are received from the service to the client applications, even under network failure or unavailability of service. Hence the service is configured with reliability features to ensure fault tolerance. From the approaches discussed in Chapter three, a conclusion can be drawn that WS-ReliableMessaging is the best standard for adding reliability features and WCF is one of the best technologies for the implementation of these reliability features. By enabling reliable sessions in the VM Web Service, fiddling through the messages, and studying the sequence of generated steps, it was determined that the service follows the WS-ReliableMessaging specification standards.

Unit testing the application ensured that the service was properly coded, and results generated from load tests demonstrated that the service could handle large numbers of simultaneous requests. The best-case and the fault-case experiments conducted to test reliability features ensured that the service was reliable and can be trusted for communication even during times of network or service failures. FIDDLER, a Web debugging proxy, displayed XML format of the communication which helped in analyzing and making sure that the reliable sessions were enabled.

This study concludes that of all the available reliability standards, WS-ReliableMessaging is the best standard in that it is easy and efficient to implement and

complements the advanced enhancements of WCF technology. The reliability of an application when implemented as a Web Service is of extreme importance. The test cases demonstrate that VM Web Service is reliable, and clients can trust the service, even during times of failures.

REFERENCES

1) Web Services overview, Eclipse documentation- Previous Release, Eclipse Galileo, http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jst.ws.doc.user/concepts/cws.html

2) Introduction to Reliable Messaging with the Windows Communication Foundation, MSDN Microsoft Library

3) FIDDLER, Web Debugging Proxy, http://www.fiddler2.com/fiddler2/

4) Douglas K. Barry, Barry Associates , Service Oriented Architecture (SOA) definition, http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jst.ws.doc.user/concepts/cws.html

5) Douglas K. Barry, Barry Associates, Service, http://www.service-architecture.com/web-services/articles/service.html

6) Web Services, Essential SOA and Web Services Resources, SearchSOA.com Definitions, http://searchsoa.techtarget.com/sDefinition/0,,sid26_gci750567,00.html

7) Douglas K. Barry, Using the Web Service Description Language, http://www.service-architecture.com/web-services/articles/web_services_explained.html

8) Web Service Description and Discovery Using UDDI Part I, MSDN Microsoft Library

9) Understanding SOAP, MSDN Microsoft library

10) Andrew Quilley, http://www.it-station.co.uk/jargon.html

11) What is Windows Communication Foundation? , MSDN Microsoft library

12) WCF, authorSTREAM, http://www.authorstream.com/Presentation/stgeorge-373691-wcf-introduction-science-technology-ppt-powerpoint/

13) Addresses, CodeIdol, http://codeidol.com/csharp/wcf/WCF-Essentials/Addresses/

14) WCF (Windows Communication Foundation) Essentials - Bindings, C# Online.NET, http://en.csharp-online.net/WCF_Essentials—Bindings

15) Introduction to Windows Communication Foundation, Contracts, Dinesh Sodani, http://beyondrelational.com/blogs/dinesh/archive/2010/07/11/introduction-to-wcf-service.aspx

16) Web Services Reliable Messaging Protocol (WS- ReliableMessaging), February 2005, http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf

17) Features of WCF, MSDN Microsoft Library

18) WSDualHttpBinding Class, MSDN Microsoft Library

19) Web Services Reliable Messaging (WS-ReliableMessaging), wsrm- 1.1-spec-cd-01, September 26th 2005

20) Software Testing, Sophisticated Functional Testing, RightHand TECHNOLOGIES, http://www.ssiembedded.com/software-testing.html

21) Unit Testing, MSDN Microsoft Library

22) Load Testing vs Performance Testing, LoadStorm, http://loadstorm.com/2010/load-testing-vs-performance-testing

23) D&D KM–IT Decommissioning and Deactivation Knowledge Management-Information Tool, http://dndkm.arc.fiu.edu/dndkm/

24) What's New in Web Services Enhancements (WSE) 3.0? MSDN Microsoft Library

25) WSE 3.0 and WS- ReliableMessaging, MSDN Microsoft Library

26) Windows Communication Foundation Architecture, MSDN Microsoft Library

27) What is IIS? , Windows IT Pro, http://www.windowsitpro.com/article/john-savills-windows-faqs/what-is-iis-.aspx

28) Neotys, Neoload, www.neotys.com