Integration Of Web Services

GURDEEP KANSAL (071265)
RAMAN KUMAR (071313)

Under the Supervision of

Mr. Yashwant Singh
Senior Lecturer
JUIT Waknaghat

Mr. Suman Saha
Senior Lecturer
JUIT Waknaghat

May – 2011

Submitted in partial fulfillment of the Degree of

BACHELOR OF TECHNOLOGY (CSE)

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN , HIMACHAL PRADESH, INDIA

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN , HIMACHAL PRADESH

CERTIFICATE

This is to certify that the work titled **"INTEGRATION OF WEB SERVICES"** submitted by **"GURDEEP KANSAL (roll no 071265)** and **RAMAN KUMAR (Roll no 071313)"** in the partial fulfillment for the award of degree of **B. Tech** (CSE) by Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	[illegible] 25/5/11
Name of Supervisor	Mr. Yashwant Singh
Designation	Senior Lecturer
Date	……………………..

ACKNOWLEDGEMENT

No venture can be completed without the blessing of Almighty. We consider it our bounded duty to bow to Almighty whose kind blessings always inspire us on the right path of life This project is our combined effort and realizes the potential of team and gives us a chance to work in co-ordination.

Science has caused many frontiers so has human efforts towards human research. Our revered guide Mr. Yashwant Singh, Senior Lecturer, Department of Computer Science and IT, JUIT, has indeed acted as a light house showing us the need of sustained effort in the field of Java and .NET to learn more and more. So we take this opportunity to thank him, for lending us stimulating suggestions, innovative quality guidance and creative thinking. He provides us the kind of strategies required for the completion of a task. We are grateful to him for the support, he provided us in doing thinks at our pace and for being patient for our mistakes.

ABSTRACT

The project entitled as **INTEGRATION OF WEB SERVICES** is a method developed to integrate two web services. Today, the ability to seamlessly exchange information between
Internal business units, customers, and partners is vital for success, yet most organizations employ a variety of disparate applications that store and exchange data in dissimilar ways and therefore cannot "talk" to one another productively. Web services have evolved as a practical, cost-effective solution for uniting information distributed between critical applications over operating system, platform, and language barriers that were previously impassable.

Web Services offer a platform neutral approach for integrating applications, so that it can be used to integrate diverse systems, in a way supported by standards rather than proprietary systems. The ability of an enterprise to have access to real-time information spanning across multiple departments, applications, platforms and systems is one of the most important driving factors behind the adoption of Web Services. Companies should first start using Web Services for their internal integration projects for business processes that are non-transactional in nature, before they venture using Web Services in B2B integration projects.

The basic idea behind this project is to design a website that has two web services integrated and designed in it and to use tools like .NET, SOAP and WDDI to implement it.

TABLE OF CONTENTS

1. Introduction

Because of the level of the application's integration, the Web services have grown in popularity and are beginning to improve the business processes. In fact, the Web services are being called the next evolution of the Web.

Web services provide a promising framework for development, integration, and interoperability of distributed software applications. Wide-scale adoption of the web services Technology in critical business applications will depend on the feasibility of building highly dependable services. Web services technology enables interaction of software components across organizational boundaries. In such distributed environment, it is critical to eliminate errors at the design stage, before the services are deployed. Web services provide a promising framework for development, integration, and interoperability of distributed software applications. Wide-scale adoption of the web services technology in critical business applications will depend on the feasibility of building highly dependable services.

The remainder of the report is structured as follows: section 2 provides information about web services architecture in conjunction with the Web services technology and the core Web services specifications; section 3 describes a case study on web services and associated key technologies. The application shows how to integrate Web Services on different platforms and how they allow the interoperability between applications running on these platforms, using the specific web services protocol stack presented in section 2. Section 4 concludes the report and presents future proposed developments.

The meaning is pretty much implicit because everybody knows what a service is, and has a more or less clear definition in mind of the Web. Maybe, that's why a lot of people talk about them without really understanding what is behind. A common example of a Web service is that of a stock quote service; there is a request for the current price of a specified stock to which the Web service responds with the price. Basically, the service receives a request, processes it, and returns a response. You could say that a simple Java servlet behaves exactly the same, but the difference is behind the scene and that's what makes the big asset Web services have over all other servlet-like components.

The client's basic needs over time don't really change, but the essential tools that are required in order to fulfill these needs are constantly evolving. Not long ago in nodes, which were present in distributed systems, existed the need to control the applications in a distributed manner. Basically if an application that was running in a node happened to go down, that application was suppose to be restarted at another node. The creation of these types of distributed applications was nearly impossible. These days, it's routine and infact there are many choices. The essential problem is not if it is possible for the components of distributed applications to communicate between them, but to choose the best technology in order to hold them together.

For example the .NET Framework, introduces good support for the two ways to architect a distributed application. Remoting is the architectural descendant of DCOM, allowing an object on one computer to make proxy-based calls to the methods of an object on another computer. Web services use a completely different technique, based on open XML and SOAP protocols, WSDL and UDDI, which are used to invoke methods on a remote machine.

XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing available services.

"Web services are dynamic programs that enable data and applications to interact with each other on the Web through ad hoc connections—without any human intervention what so ever", said Sidharth, technical product manager for identity management at Sun. A Web service is normally intended to be a distributed application component. Its clients are other applications, not human beings. A Web Service is any piece of code that can communicate with other pieces of code via common Internet technology. A Web Service is a "virtual component" that hides "middleware idiosyncrasies" like the underlying component model, invocation protocol as far as possible.

The main advantages of Web services are flexibility and versatility: they support a lot of architecture and are independent of platforms and designs. Web services are built on several technologies that work in conjunction with emerging standards to ensure security and manageability, and to ensure that Web services can be combined to work independent of a vendor. Also Web services win on ease of development and interoperability.

Web services distributed computing model allows application-to-application communication. For example, one purchase-and-ordering application could communicate to an inventory application that specifies the items that need to be reordered or a Web service from a credit bureau which requests the credit history from the loan services, for prospective borrowers. In both cases, the data interaction must be protected to preserve its confidentiality. Web Services are considered to be the future of the Internet. They are independent of the platform and also of the technology, but in reality they are XML/SML collections of standards which allow the interaction between systems (programs). Heather Kreger, one of the IBM's lead architects for SOA Standards which developed the standards for Web services, thought that Web Services are like an interface which describes a collection of operations, network accessible throughout the XML standard messages. Web Services have the main role to access different services and different data from different machines, and so they offer to the clients a single public interface. First advantage, the format of the messages exchanged between a client and a Web service is specified by a standard called SOAP. This means that Web services are not platform dependant. You could have, for example, a Web service written in c# and deployed on the .NET Framework, communicating with a Java client running on a Linux; all this can happen because they both speak the same language: SOAP. Then, every Web service is deployed with a WSDL file that acts like an instruction manual; in case someone wants to use a particular Web Service, he simply has to look at that WSDL file to learn how to communicate with the corresponding service and use it. Finally, you can find a specific Web service using UDDI; that's the yellow pages of Web services where you can search by different criteria to find the appropriate service and its access points (URL).

All this is simply an exchange of XML documents over HTTP. Now that we have the big picture, let's take a closer look at those different components..

1.1 XML

XML is the acronym for Hypertext Transfer Protocol. HTTP is a W3C standard defined as an application-level protocol for distributed, collaborative, hypermedia information systems that can be built independently of the data being transferred. HTTP has been in use by the World-Wide Web global information initiative since 1990. Thus, we can say that HTTP is the most popular transfer protocol and it's supported on almost all platforms. By implementing this standard, combined with XML, Web services remove almost all frontiers between platforms.
More information about HTTP can be found on the W3C website .

It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.
The following example is a note to Tove, from Jani, stored as XML:

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

That is because the XML language has no predefined tags.

The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.).

XML allows the author to define his/her own tags and his/her own document structure.

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store, and transport information. XML's performance impact doesn't merely affect the network; larger messages mean more CPU time to parse and un-parse messages, increased storage requirements (here the tag-to-text ratio is very important) and increased memory usage (especially if DOM-based parsing is used).
Similarly, a Web Services interface should emerge from business requirements, not from a decision to take an existing interface and "expose it" as a Web Service. The key to accomplishing the above is through the use of patterns and guidelines, which will help manage the complexity and decrease the risk of adoption of XML and Web Services. Several industry patterns, such as Multichannel Interface, Spoke and Hub, Canonical Data, etc., are depicted above.
Focus on identifying patterns in the design and application strategy. The note above is quite self descriptive. It has sender and receiver information, it also has a heading and a message body.

But still, this XML document does not DO anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

1.2 HTTP

HTTP is the acronym for Hypertext Transfer Protocol. HTTP is a W3C standard defined as an application-level protocol for distributed, collaborative, hypermedia information systems that can be built independently of the data being transferred. HTTP has been in use by the World-Wide Web global information initiative since 1990. Thus, we can say that HTTP is the most popular transfer protocol and it's supported on almost all platforms. By implementing this standard, combined with XML, Web services remove almost all frontiers between platforms.

HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed.

HTTP is called a stateless protocol because each command is executed independently, without any knowledge of the commands that came before it. This is the main reason that it is difficult to implement Web sites that react intelligently to user input. This shortcoming of HTTP is being addressed in a number of new technologies, including ActiveX, Java, JavaScript and cookies.

HTTP functions as a request-response protocol in the client-server computing model. In HTTP, a web browser, for example, acts as a client, while an application running on a computer hosting a web site functions as a server. The client submits an HTTP request message to the server. The server, which stores content, or provides resources, such as HTML files, or performs other functions on behalf of the client, returns a response message to the client. A response contains completion status information about the request and may contain any content requested by the client in its message body.

A client is often referred to as a user agent (UA). As well as web browsers, web crawlers are another common user agent. These include the indexing software used by search providers. Voice browsers are another less common but important class of user agent.

The HTTP protocol is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of the original, so-called origin server to improve response time. HTTP proxy servers at network boundaries facilitate communication when clients without a globally routable address are located in private networks by relaying the requests and responses between clients and servers.

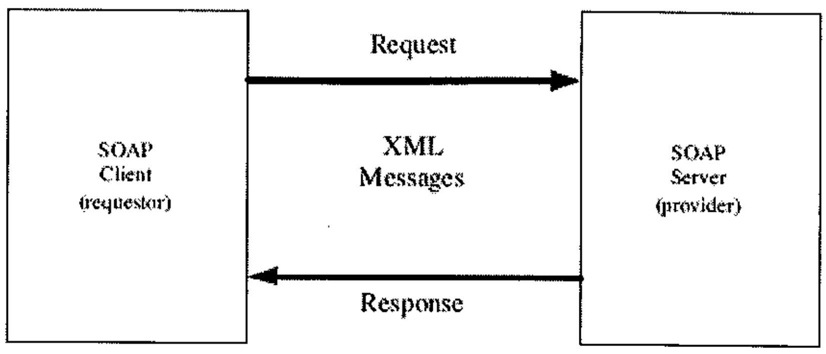
HTTP is an Application Layer protocol designed within the framework of the Internet Protocol Suite. The protocol definitions presume a reliable Transport Layer protocol for host-to-host data transfer. The Transmission Control Protocol (TCP) is the dominant protocol in use for this purpose. However, HTTP has found application even with unreliable protocols, such as the User Datagram Protocol (UDP) in methods such as the Simple Service Discovery Protocol (SSDP).

HTTP Resources are identified and located on the network by Uniform Resource Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the http or https URI schemes. URIs and the Hypertext Markup Language (HTML),

form a system of inter-linked resources, called hypertext documents, on the Internet, that led to the establishment of the World Wide Web in 1990 by English physicist Tim Berners-Lee.
The original version of HTTP (HTTP/1.0) was revised in HTTP/1.1. HTTP/1.0 uses a separate connection to the same server for every request-response transaction, while HTTP/1.1 can reuse a connection multiple times, to download, for instance, images for a just delivered page. Hence HTTP/1.1 communications experience less latency as the establishment of TCP connections presents considerable overhead.
An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request. It establishes a Transmission Control Protocol (TCP) connection to a particular port on a host (typically port 80; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested resource, an error message, or some other information.

1.3 SOAP

SOAP is the acronym for **Simple Object Access Protocol**. The role of SOAP is to encode the data in XLM format and to make the message exchange possible between the applications in XML format. It uses the model request answer, where the request is placed by the SOAP client, and the answer is given by the service provider, named SOAP server. Everything is shown in the below situated Figure.



SOAP's basic request-response model

The protocol is used both to send and to receive messages from the Web Service.

One advantage is to encapsulate the functionality of the RPC (Remote Procedure Call) using the extensibility and the functionality of the XML. SOAP defines a format for both messages, and a model for their processing by the receiver. In addition, SOAP – may also define a framework for protocol links,

so that the SOAP messages can be transferred using the protocol stack from the transport level.

A SOAP message consists of a SOAP envelope, the root of the message, which in turn contains an optional header, and, necessarily, a body, independent of each other.

SOAP message passes on its way from sender to receiver through many SOAP nodes, which can change the message. All the SOAP nodes form SOAP message path.

The Header contains general information about security – authentication and session, and about the message processing by the intermediary nodes. The data regarding the authentication usually is encrypted using WS-Security standard. The tag - "body" never misses from a SOAP message. Most of the times it is the last child of the "Envelope" node and it contains the information that is going to be transferred between applications (Web service input or output). It is shown an example of a SOAP message.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <GetMaterialsResponse xmlns="http://tempuri.org/">
      <GetMaterialsResult>xmlxml</GetMaterialsResult>
    </GetMaterialsResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP Message

In the above example, in the SOAP envelope the XML namespace and the type of the used message encoding are not specified. The Header node is missing in this example, and the body node

contains the result of a *"GetMaterials"* method, which is a serialized Data Table type object in XML format.

SOAP was originally an acronym for Simple Object Access Protocol, But since SOAP Version 1.2 (SOAP 1.2 Part 0, 2003; SOAP 1.2 Part 1, 2003) it is technically no longer an acronym.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://alex/MSEWS"
```

```
xmlns:types="http://alex/MSEWS/encodedTypes"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encodi
ng/">
<tns:GetAvailableSEWS />
</soap:Body>
</soap:Envelope>
```

<u>SOAP request to the MSEWS, invoking the GetAvailableSEWS function.</u>

SOAP is an XML-based protocol from the W3C for exchanging data over HTTP.

It provides a simple, standards-based method for sending XML messages between applications. Web services use SOAP to send messages between a service and its client(s). Because HTTP is supported by all Web servers and browsers, SOAP messages can be sent between applications regardless of their platform or programming language. This quality gives Web services their characteristic interoperability.

SOAP messages are XML documents that contain some or all of the following elements:

Envelope – specifies that the XML document is a SOAP message; encloses the message itself.

Header (optional) – contains information relevant to the message, e.g., the date the message was sent, authentication data, etc.

Body – includes the message payload.

Fault (optional) – carries information about a client or server error within a SOAP message.

Data is sent between the client(s) and the Web service using request and response SOAP messages, the format for which is specified in the WSDL definition. Because the client and server adhere to the WSDL contract when creating SOAP messages, the messages are guaranteed to be compatible.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://alex/MSEWS"
xmlns:types="http://alex/MSEWS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encodi
ng/">
<types:GetAvailableSEWSResponse>
<GetAvailableSEWSResult
xsi:type="xsd:string">&lt;?xml             version="1.0"
encoding="utf-16" standalone="yes"?&gt;
&lt;availableSEWS&gt;
&lt;SEWS&gt;


&lt;name&gt;altavista.java&lt;/name&gt;
&lt;binding&gt;http://localhost:8080/SEWSaltavista/SEWS?
WSDL&lt;/binding&gt;
&lt;/SEWS&gt;
&lt;/availableSEWS&gt;</GetAvailableSEWSResult>
</types:GetAvailableSEWSResponse>
</soap:Body>
</soap:Envelope>
```

SOAP response from the MSEWS

1.4 WSDL

Web Service Description Language (WSDL) defines an XML grammar for describing network services as a set of endpoints that accept messages containing either document-oriented (style=”document”) or procedure-oriented (style=”rpc”) information. The endpoint is defined by a network protocol and a message format, however, the extensible characteristic of WSDL allow the messages and endpoints being described regardless of what message formats or network protocols are being used to communicate. In other words, a WSDL file is an XML document that describes a set of SOAP messages and how the messages are exchanged. Some very good tools for WSDL file processing can be found on Internet [10]; going from verification of WSDL files to automatic generation of proxy classes or SOAP request/response messages out of them.

To illustrate all this, please refer to LISTING 2-3 the WSDL document of the altavista Search Engine Web Service (SEWS) described in details in the next chapter. You can distinguish five distinct parts of a WSDL file on that example: <types>, <message>, <portType>, <binding> and <service> that are linked together.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://alex/SEWS"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
```

```
targetNamespace="http://alex/SEWS"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types />
<message name="SearchForSoapIn">
<part name="String_1" type="s:string" />
<part name="int_2" type="s:int" />
</message>
<message name="SearchForSoapOut">
<part name="SearchForResult" type="s:string" />
</message>
<portType name="SEWSSoapPort">
<operation name="SearchFor">
<input message="tns:SearchForSoapIn" />
<output message="tns:SearchForSoapOut" />
</operation>
</portType>
<binding name="SEWSSoapPort" type="tns:SEWSSoapPort">
<soap:binding       transport="http://schemas.xmlsoap.org/soap/http"
style="rpc" />
<operation name="SearchFor">
<soap:operation            soapAction="http://alex/SEWS/SearchFor"
style="rpc" />
<input>
<soap:body       use="encoded"       namespace="http://alex/SEWS"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
<output>
<soap:body       use="encoded"       namespace="http://alex/SEWS"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
</binding>
<service name="SEWS">
<port name="SEWSSoapPort" binding="tns:SEWSSoapPort">
<soap:address
location="http://localhost/SEWSaltavista/SEWS.asmx" />
</port>
</service>
</definitions>
```

WSDL document of the altavista SEWS

The description level, located above the packing level, is represented by the WSDL protocol, being based on the XML standard. WSDL is a language written in XML, used as a model for describing Web services. WSDL reached version 2.0, but in version 1.1 the **D** stood for Definition. Version
1.2 of WSDL was renamed WSDL 2.0 because of the major differences between the two versions.

New features in WSDL 2.0

Nowadays, W3C recommends using WSDL 2.0 [11], but the problem is that it is not fully supported in all developing environments. The main differences between the two versions are:
• in WSDL 2.0 there's binding to all the HTTP request methods, whereas in WSDL 1.1 only the GET and POST methods;
• in WSDL 2.0 further semantics were added to the description language;
• WSDL 2.0 offers better support for RESTful web services;
• renaming of PortTypes (WSDL 1.1) into Interfaces (WSDL 2.0);
• renaming of Ports (WSDL 1.1) into Endpoints (WSDL 2.0);
• WSDL 2.0 can be implemented in a much simpler way.
WSDL is used in combination with SOAP and the XML schema representing the web service description. The main purpose of WSDL is that it leverages the connection between a client program and a web service, by determining the server available operations.

WSDL Components

Port/Endpoint – defines the address or connection to a web service; usually, it is represented by a simple URL.
Service – consists of a set of ports/endpoints, meaning the system functions exposed to the web based protocols.
Binding – defines a concrete message format and transmission protocol which may be used to define a port/endpoint.
PortType/Interface – defines a web service, all the operations that can be performed, and the messages used to perform the operation.
Operation – is an interaction with the service (a method) formed by a set of messages exchanged between the service and the other programs involved in the interaction.
Type – describes the data type definitions that are relevant for the exchanged messages. The endpoint is defined by a network protocol and a message format, however, the extensible characteristic of WSDL allow the messages and endpoints being described regardless of what message formats or network protocols are being used to communicate.
Components 1-3 represent the concrete section of a WSDL, and components 4-6 represent the abstract section.

The next figure shows an example of a WSDL message

```
- <definitions name="ServiceSAP" targetNamespace="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx?WSDL">
  - <wsdl:message name="GetMaterialsSoapIn">
      <wsdl:part name="parameters" element="tns:GetMaterials"/>
    </wsdl:message>
  - <wsdl:message name="GetMaterialsSoapOut">
      <wsdl:part name="parameters" element="tns:GetMaterialsResponse"/>
    </wsdl:message>
  - <wsdl:portType name="ServiceSAPSoap">
    - <wsdl:operation name="GetMaterials">
        <wsdl:input message="tns:GetMaterialsSoapIn"/>
        <wsdl:output message="tns:GetMaterialsSoapOut"/>
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="ServiceSAPSoap" type="tns:ServiceSAPSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    - <wsdl:operation name="GetMaterials">
        <soap:operation soapAction="http://tempuri.org/GetMaterials" style="document"/>
      - <wsdl:input>
          <soap:body use="literal"/>
        </wsdl:input>
      - <wsdl:output>
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="ServiceSAP">
    - <wsdl:port name="ServiceSAPSoap" binding="tns:ServiceSAPSoap">
        <soap:address location="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx"/>
      </wsdl:port>
    </wsdl:service>
```

WSDL is maintained by the W3C, WSDL is an XML-based format for describing Web services. Clients wishing to access a Web service can read and interpret its WSDL file to learn about the location of the service and its available operations. In this way, the WSDL definition acts as the initial Web service interface, providing clients with all the information they need to interact with the service in a standards-based way. Through the WSDL, a Web services client learns where a service can be accessed, what operations the service performs, the communication protocols the service supports, and the correct format for sending messages to the service.

A WSDL file is an XML document that describes a Web service using six main elements:

Port type – groups and describes the operations performed by the service through the defined interface.

Port – specifies an address for a binding, i.e., defines a communication port.

Message – describes the names and format of the messages supported by the service.

Types – defines the data types (as defined in an XML Schema) used by the service for sending messages between the client and server.

Binding – defines the communication protocols supported by the operations provided by the service.

Service – specifies the address (URL) for accessing the service.

The WSDL document that describes a Web service acts as a contract between Web service client and server. By adhering to this contact the service provider and consumer are able to exchange data in a standard way, regardless of the underlying platforms and applications on which they are operating.

WSDL builds on XML Schema by making it possible to fully describe Web services in terms of messages, operations, interfaces (portTypes), bindings, and service endpoints. WSDL definitions make it possible to generate code that implements the given interface, on either the client or the server, making Web services accessible to the masses.

1.5 UDDI

Universal Discovery Description and Integration (UDDI) is the yellow pages of Web services. A UDDI directory entry is an XML file that describes a business and the services it offers. Both can be categorized and have keys so one can find a provider or a service by different ways. Web services are also defined through a document called a Type Model (tModel) that describes their interface. A tModel is simply a WSDL file without the <service> tag; it contains information to generate the different proxy classes or SOAP messages to communicate with the Web service but it does not specify the access point of the service.

Some public UDDI Business Registry Nodes, where you can publish your own Web services or search for existing ones, are available on the Internet. If you do not want to distribute openly your services, you can install your own UDDI server. Independently of the choice, there are two ways of accessing a UDDI registry. The first one is through a web browser interface that provides form-based access to register or search for a Web service. The second one is programmatically using standardized API's; the main advantage of that solution is that it can be used dynamically at runtime. UDDI is a standard sponsored by OASIS (Organization for the Advancement of Structured Information Standards). Often described as the yellow pages of

Web services, UDDI is a specification for creating an XML-based registry that lists information about businesses and the Web services they offer. UDDI provides businesses a uniform way of listing their services and discovering services offered by other organizations. Though implementations vary, UDDI often describes services using WSDL and communicates via SOAP messaging.

Registering a Web service in a UDDI registry is an optional step, and UDDI registries can be public or private (i.e. isolated behind a corporate firewall).

To search for a Web service, a developer can query a UDDI registry to obtain the WSDL for the service he/she wishes to utilize. Developers can also design their Web services clients to receive automatic updates about any changes to a service from the UDDI registry.

UDDI stores information about web services, it consists of web services interfaces written in WSDL. UDDI can be interrogated via SOAP messages and provides access to WSDL documents describing certain protocol bindings and message formats to interact with the web services. UDDI terminology contains also the following:

• Nodes – servers which support UDDI specifications nodes belong to a registry;

• Registries – collections of one or more UDDI nodes.

Benefits of UDDI

All businesses can benefit of UDDI because it solves the following problems:

• Discovering the right business from millions of online businesses;

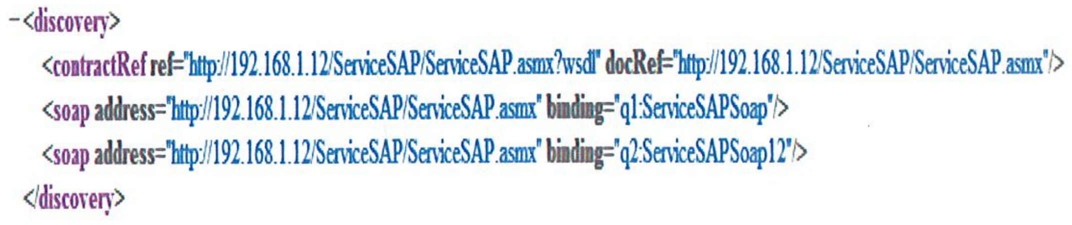
• Once the preferred business is discovered, UDDI enables how to enable commerce;

• New customers can be reached and access to current customers can be increased;

• Market reach and offerings can be expanded;

• Barriers are removed to allow rapid participation in the global Internet economy;

• Services and business processes are programmatically described in a single, open, and secure environment.



```
-<discovery>
   <contractRef ref="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx?wsdl" docRef="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx"/>
   <soap address="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx" binding="q1:ServiceSAPSoap"/>
   <soap address="http://192.168.1.12/ServiceSAP/ServiceSAP.asmx" binding="q2:ServiceSAPSoap12"/>
 </discovery>
```

it is shown an example of a UDDI message

Although the basic specifications of Web Services: XML, SOAP, WSDL and UDDI provide an acceptable level of interoperability and integrity a significant effort has made to increase the applications area of Web Services, and to address to higher various issues from the real world. Thus new specifications emerged for Web Services' reliability, security, metadata management, transactions and orchestration, all of which have extended the Web Services' architecture. Among the new specifications it is worth to be mentioned:

• Metadata Management: WS-Addressing, WS-Policy, WS-MetadataExchange;

• Reliable Messaging: WS-Reliability, WS-ReliableMessaging, WS-Eventing, WS-Notifications;

• Security: WS-Authorization, WS-SecurityPolicy, WS-Trust, WSSecureConversation, WS-Federation, WS-Privacy, XML Encryption, XML Signature;

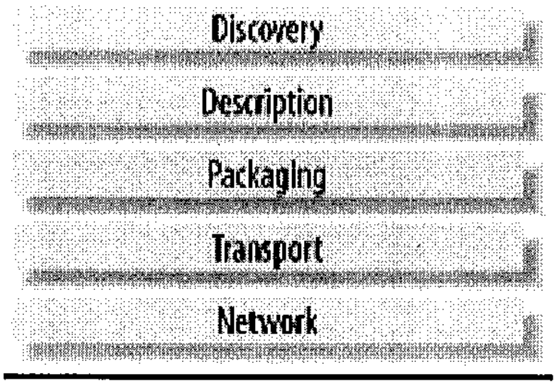
• Transactions: WS-Transactions family(WS-AtomicTransaction, WSBusinessActivity, WS-Coordination), WS-Composite Application Framework(WS-CAF);

• Orchestration

• Choreography

2.1 The Web Services Architecture

The figure below describes the architecture of the Web Services. The architecture of the Web Services resembling with the TCP/IP reference model is presented in five levels: Network, Transport, Packing, Description and Discovery. Each level is represented by different basic protocols. The network level concurs with the network level from the TCP/IP reference model, offering basic communication, addressing and rooting. Above the network level there is the transport level that offers the opportunity of direct communication between the existing applications from the network. The most important protocols are TCP/IP, UDP, FTP, HTTP, SMTP, Jabber.



The web services can be implemented above any of the other protocols. The Packing level, which is above the Transport level, "packs" the data in the XML format – a format known by all the other parties involved in communication. XML and SOAP –

Simple Object Access Protocol, are basic protocols of the Packing Level and are produced by the W3C standard.

2.2 Web Services Benefits

Web services provide several technological and business benefits, a few of which include:

- **Application and data integration**
- **Versatility**
- **Code reuse**
- **Cost savings**

The inherent interoperability that comes with using vendor, platform, and language independent XML technologies and the ubiquitous HTTP as a transport mean that any application can communicate with any other application using Web services. The client only requires the WSDL definition to effectively exchange data with the service – and neither part needs to know how the other is implemented or in what format its underlying data is stored. These benefits allow organizations to integrate disparate applications and data formats with relative ease.

Web services are also versatile by design. They can be accessed by humans via a Web-based client interface, or they can be accessed by other applications and other Web services. A client can even combine data from multiple Web services to, for instance, present a user with an application to update sales, shipping, and ERP systems from one unified interface – even if the systems themselves are incompatible. Because the systems exchange information via Web services, a change to the sales database, for example, will not affect the service itself.

Code re-use is another positive side-effect of Web services' interoperability and flexibility. One service might be utilized by several clients, all of which employ the operations provided to fulfil different business objectives. Instead of having to create a custom service for each unique requirement, portions of a service are simply re-used as necessary.

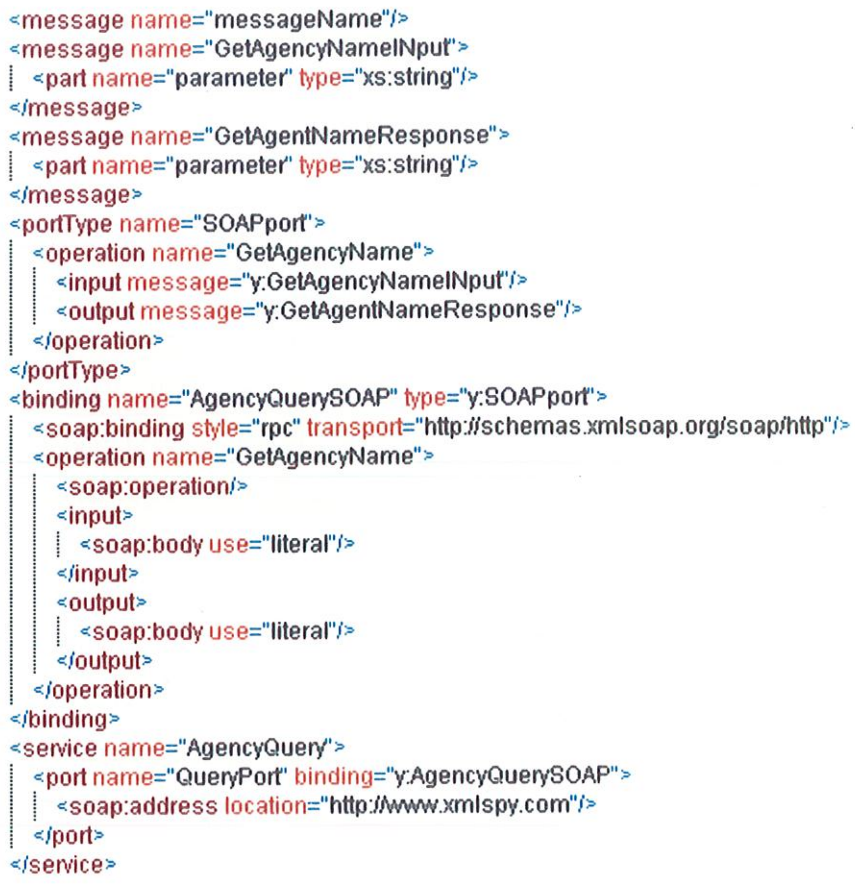
All these benefits add up to significant cost savings. Easy interoperability means the need to create highly customized applications for integrating data, which can be expensive, is removed. Existing investments in systems development and infrastructure can be utilized easily and combined to add additional value. Since Web services are based on open standards their cost is low and the associated learning curve is smaller than that of many proprietary solutions.

Finally, Web services take advantage of ubiquitous protocols and the Web infrastructure that already exists in every organization, so they require little if any additional technology investment.

2.3 Web services development challenges

With the numerous advantages of Web services come a few challenges. Most significantly, though Web services themselves are designed to be simple, actually developing and implementing them can be complex. WSDL syntax becomes complicated quickly, especially when building a service with multiple operations in a text-based editor. A snippet of WSDL code is shown in Figure below.
Even looking at the completed code, it's difficult to follow the chain of connections from a service name, to the binding, to the

port type, and so on, never mind writing the code correctly by hand.



```
<message name="messageName"/>
<message name="GetAgencyNameINput">
    <part name="parameter" type="xs:string"/>
</message>
<message name="GetAgentNameResponse">
    <part name="parameter" type="xs:string"/>
</message>
<portType name="SOAPport">
    <operation name="GetAgencyName">
        <input message="y:GetAgencyNameINput"/>
        <output message="y:GetAgentNameResponse"/>
    </operation>
</portType>
<binding name="AgencyQuerySOAP" type="y:SOAPport">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetAgencyName">
        <soap:operation/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
<service name="AgencyQuery">
    <port name="QueryPort" binding="y:AgencyQuerySOAP">
        <soap:address location="http://www.xmlspy.com"/>
    </port>
</service>
```

There are tools that will auto-generate WSDL code based on an existing application that a developer wants to expose as a Web service. However, best practices dictate that designing the WSDL be the first step in architecting Web services. The contract-first approach to Web services development has many advantages. Designing the interface first results in better overall planning prior to implementing the service and helps ensure the service will be effective in multiple client scenarios. In addition, WSDL uses language-independent XML Schema for type definitions, allowing

Web service designers to specify strongly-typed requirements for communicating with the Web service. Using XML Schema, a Web services developer can specify that an email address must follow a certain pattern (username@domainname.xxx), that a product quantity must be a positive whole integer, that a first name and last name are required input values, and so on. This same level of data typing is not provided when a WSDL is auto-generated from an existing Java or C# application.

In addition, because WSDL is standards-based, designing the WSDL first, then building the Web service based on this definition prevents developers from including language-specific types and constructs in their Web service.

This ensures that any Web services client can interact with the service without interoperability issues. Though the WSDL contract-first approach may seem more rigorous at first, the resulting benefits make the service more effective by ensuring interoperability – which, after all, is the rationale behind using Web services in the first place. Lack of tool support is often cited as the biggest obstacle to the contract-first approach to Web services design.

Another challenge arises after the WSDL is defined, when the developer must actually write the Java or C# code to connect the required data sources and implement the service on a server. Given that even the simplest of Web services may require thousands of lines of code, this process is often time consuming and error prone. In addition, given that Web services require the expertise of XML developers who may not necessarily also be Java or C# experts, this step can be especially challenging.

Recognizing these barriers to Web services development, We have created a suite of tools for designing and building Web services in a graphical manner.

This approach allows developers to build well designed, standards-conformant, interoperable Web services – without manual coding.

2.4 "Trip Planner" Web services development

This unique graphical approach simplifies WSDL development by enabling an easy-to-understand visual process.
Instead of working solely with a text view, developers can build their WSDL files graphically, with full validation and editing help, and the corresponding WSDL code is generated behind the scenes where it can be referenced and edited at any time. This eliminates much of the complexity otherwise associated with the design-before-implementation development approach.
Once a WSDL file is created, the Web services developer must still write the extensive Java or C# code to programmatically connect the operations defined in the WSDL with the data they will return. Altova MapForce automates this step with its graphical WSDL mapping interface and code generation capabilities. MapForce displays WSDL operations as visual mapping items that can be dragged and dropped to connect with their corresponding data sources. You can also filter and process data before returning it as a Web services response message. Once a WSDL mapping design is complete, MapForce auto-generates the Java or C# code required to implement the service server-side.
Completing the Web services design loop are the SOAP features in XMLSpy.
XMLSpy includes a SOAP client that automatically creates SOAP messages based on criteria defined in a WSDL document. Then it

actually sends the message to the Web service and displays the response. This feature is very useful for testing your own Web services implementations, and it's invaluable for interpreting WSDL documents when you need to create a client based on someone else's WSDL definition.

XMLSpy also includes a SOAP debugger, which acts as a Web services proxy between the client and server, allowing you to intercept and examine the actual messages sent between a client and server. The SOAP debugger allows you to set functional or conditional breakpoints on request and response messages to quickly track down and debug any problems with aWeb service client or server implementation.

3. Web Services Frameworks

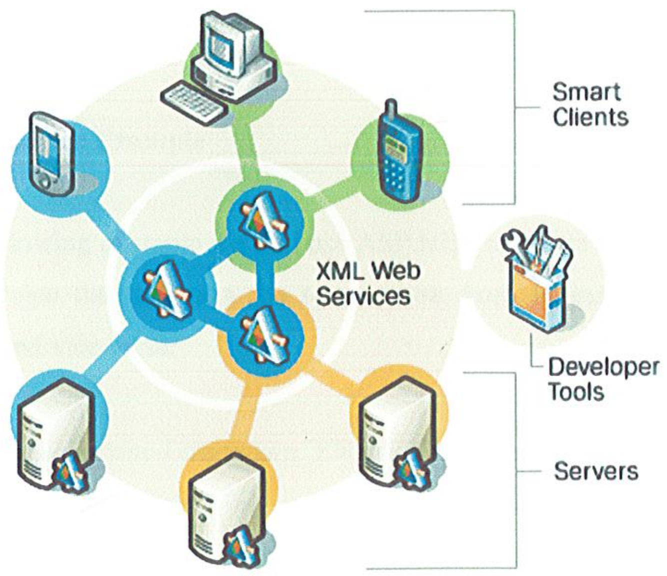
One of the mains goals of this thesis is to show that 'Web services' is not just another buzzword spinning around the Internet. The objective is to demonstrate, that beyond theories and concepts, it is possible to build Web services and real applications that integrate them. My intention is to exploit most of the features offered by Web services through the implementation of a Trip Planner Web Service on two different platforms: Java and .NET.

If someone is still skeptic as far as the real added value of the Web services, he may change his opinion, after reading this document, and understand the reasons why companies like Microsoft re-built their entire strategy around Web service.

3.1 .NET

Also known as Microsoft® .NET, is defined as set of software technologies designed to connect your world of information, people, systems, and devices. It enables a high level of software integration through the use of Web services.

The future of Web services, dot-NET and J2EE may be unclear, but the subject has already generated intense debate and controversy. Issues range from privacy concerns to debates on standards for conducting international business over the Internet. Business considerations aside, Microsoft dot-NET contains some cool networking technology that is worthy of a closer look.



Web services are the core of .NET; it's the glue that holds and connects everything together (see FIGURE).

To be able to build, deploy and run Web services, Microsoft® has put in place the .NET Framework. The same environment is also used for building other Web-based and smart client applications. It includes the common language runtime, class libraries and a multiple-language support that enables developers to use the appropriate language for a specific component with the capacity to combine them within a single application.
The Microsoft® .NET Framework Software Development Kit (SDK) including the .NET Framework, as well as everything you need to write, build, test, and deploy Microsoft® .NET–connected applications and technologies, can be downloaded at http://msdn.microsoft.com/netframework/downloads/.
Microsoft® Internet Information Services (IIS) is required to be able to run your Web services. Besides, Visual Studio .NET can optionally be installed for rapidly building and integrating Web services.

A Web Service Example

In the following example we will use ASP.NET to create a simple Web Service that converts the temperature from Fahrenheit to Celsius, and vice versa:

```
<%@WebService Language="VBScript"
Class="TempConvert" %>

Imports System
Imports System.Web.Services

Public Class TempConvert :Inherits WebService
```

```
<WebMethod()> Public Function FahrenheitToCelsius
(ByVal Fahrenheit As String) As String
  dim fahr
  fahr=trim(replace(Fahrenheit,",","."))
  if fahr="" or IsNumeric(fahr)=false then return "Error"
  return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit
(ByVal Celsius As String) As String
  dim cel
  cel=trim(replace(Celsius,",","."))
  if cel="" or IsNumeric(cel)=false then return "Error"
  return ((((cel) * 9) / 5) + 32)
end function
end class
```

This document is saved as an .asmx file. This is the ASP.NET file extension for XML Web Services.

Example Explained

Note: To run this example, you will need a .NET server.
The first line in the example states that this is a Web Service, written in VBScript, and has the class name "TempConvert":

```
<%@ WebService Language="VBScript"
Class="TempConvert" %>
```

The next lines import the namespace "System.Web.Services" from the .NET framework:

```
Imports System
Imports System.Web.Services
```

The next line defines that the "TempConvert" class is a WebService class type:

```
Public Class TempConvert :Inherits WebService
```

The next steps are basic VB programming. This application has two functions. One to convert from Fahrenheit to Celsius, and one to convert from Celsius to Fahrenheit.

The only difference from a normal application is that this function is defined as a "WebMethod()".
We could use "WebMethod()" to convert the functions in your application into web services:

```
<WebMethod()> Public Function FahrenheitToCelsius
(ByVal Fahrenheit As String) As String
 dim fahr
 fahr=trim(replace(Fahrenheit,",","."))
 if fahr="" or IsNumeric(fahr)=false then return "Error"
 return ((((fahr) - 32) / 9) * 5)
end function

<WebMethod()> Public Function CelsiusToFahrenheit
(ByVal Celsius As String) As String
```

```
dim cel
cel=trim(replace(Celsius,",","."))
if cel="" or IsNumeric(cel)=false then return "Error"
return ((((cel) * 9) / 5) + 32)
end function
```

Then, end the class:

```
end class
```

Publish the .asmx file on a server with .NET support, and you will have your first working Web Service.
Look at our example Web Service

ASP.NET Automates the Process

With ASP.NET, you do not have to write your own WSDL and SOAP documents.
If you look closer at our example Web Service, you will see that ASP.NET has automatically created a WSDL and SOAP request.
However, just as we use frameworks such as ASP and ASP.NET to build Web applications, we would much rather use a framework for building Web Services.
The reasoning is quite logical. We don't need to reinvent the plumbing—that is, at a high level, the capability to serialize our data as XML, transport the data using HTTP, and de-serialize the XML back to meaningful data. Instead, we want a framework that makes building Web Services easy, allowing us to focus on the application logic not the plumbing. ASP.NET provides this framework for us.

From a developer's point of view, if you have ever written application logic, you have the required skills to author ASP.NET

Web Services. More importantly, if you're at all familiar with ASP or ASP.NET application services, (application state memory, and so on) you can also leverage these skills when you build ASP.NET Web Services.

a Simple Example

```
Public Class MyMath
 Public Function Add(a As Integer, b As Integer) As Integer
  Return a + b
 End Function
End Class
```

We could use this class and its method as follows:

```
Dim mymath As new MyMath
Dim result As Integer
result = mymath.Add(10, 20)
```

To expose the above class, MyMath, as an ASP.NET Web Service we need to move the application logic into a *.asmx file. Just as we use the extension *.aspx for ASP.NET Pages, we use *.asmx to tell ASP.NET that the file is an ASP.NET Web Service.

After we created the *.asmx source file and add our application logic, we need to make a few more small changes:

```
<%@ WebService Language="VB" Class="MyMath" %>
Public Class MyMath
 Public Function <WebMethod()>Add(a As Integer, b As
Integer) As Integer
  Return a + b
 End Function
End Class
```

Changes to our source

The changes we've made to the *.asmx file include adding a **WebService** directive that names both the Language as well as the Class we're exposing as a Web Service. The **WebService**directive is required, as we must tell ASP.NET the class that contains the application logic. Next, we've added a **<WebMethod()>** attribute to our Add() function declaration. An attribute is a declarative code element that lets us change the behavior of our application logic without necessarily writing more code. In the case of the **<WebMethod()>** attribute, this tells ASP.NET that the method with this attribute is to be treated as 'Web callable'. Web callable in the sense that ASP.NET does the necessary work for this method to support SOAP.

3.2 JAVA

Sun couldn't let all the Java developers without a solution to build, deploy and run Web service so they released the Java Web Service Developer Pack (WSDP). Based on the Java 2 SDK, this toolset adds new API including XML Messaging (JAXM), XML Processing (JAXP), XML Registries (JAXR), XML-based RPC (JAX-RPC) and the SOAP with Attachments (SAAJ). It comes also with the Apache Tomcat container and the famous Ant Build Tool; therefore, all you need to build your first Web service is your favorite text editor.

The main advantage Java WSDP has over the .NET Framework is that it is supported not only on the Windows® platforms but also on Solaris™ 2.9 and RedHat Linux® 7.2.

More information about the Java WSDP can be found on the Java website.

Other tools exist on the market for the development of Web services using Java; the latest version of JBuilder includes the Borland Web Services Kit supporting the JAX-RPC standards and the Apache Axis project. JBuilder is, like Visual Studio .NET, a development tool but with the benefit of being cross-platform (available on Windows®, Solaris™, Linux® and Max® OS X). JBuilder can be downloaded.

Another example is GLUE, a Web Services platform that allows you to publish any Java object as a Web service without modification. With GLUE, any Web service can also be accessed as if it were a local Java object (no stub generators or command line tools are necessary).

To use standard Java for creating a webservice the following steps are done

- create a standard java class
- use annotations to describe the class as a webservice
- run a command line tool to create the WSDL
- you run the class for example via a main program

EXAMPLE-

The server Java project

Create a Java project "de.vogella.webservice.java6.first.provider". Create a package with the same name and then the following class.

```
package
de.vogella.webservice.java6.first.provider;


import javax.jws.WebService;

import javax.jws.soap.SOAPBinding;

import javax.jws.soap.SOAPBinding.Style;

import javax.xml.ws.Endpoint;


@WebService

public class WiseQuoteServer {

      @SOAPBinding(style = Style.RPC)

      public String getQuote(String category) {

            if (category.equals("fun")) {

                  return "5 is a sufficient
approximation of infinity.";

            }

            if (category.equals("work")) {

                  return "Remember to enjoy life,
even during difficult situatons.";

            } else {

                  return "Becoming a master is
relatively easily. Do something well and then
continue to do it for the next 20 years";

            }

      }


      public static void main(String[] args) {

            WiseQuoteServer server = new
WiseQuoteServer();
```

```
                Endpoint endpoint = Endpoint.publish(

        "http://localhost:9191/wisequotes",
server);
```

If you start your main program the service should be up and running. Using the console switch to an empty directory and call the wsimport command line tool which is part of the JDK.

Using the Webservice

Create a new Java project "de.vogella.webservice.java6.first.consumer" create a package with the same name, copy the created java classes to your new project / package and adjust the path name.

Create a new Java class TestWS.java with the following coding. The coding demonstrates how to get the connection directly and how to get the connection with specifying the URL. Specifying the URL is useful in case you what to use a TCP-monitor to trace the TCP data. In this case you would direct the client to the TCP monitor which would re-direct the TCP data to the server and vice versa.

```
package de.vogella.webservice.java6.first.consumer;


import java.net.MalformedURLException;

import java.net.URL;


import javax.xml.namespace.QName;


public class TestWS {

        public static void main(String[] args) {
```

```
			// Settting up the server connection

			WiseQuoteServerService    service    =    new
WiseQuoteServerService();

			WiseQuoteServer          servicePort          =
service.getWiseQuoteServerPort();

			// Calling the webservice

		System.out.println(servicePort.getQuote("fun"));

		System.out.println(servicePort.getQuote("work"));


			// Alternatively if you want to specific the URL
directly

			try {

				URL          url          =          new
URL("http://localhost:9191/wisequotes?wsdl");

				WiseQuoteServerService
serviceWithUrl = new WiseQuoteServerService(

							url,

							new QName(

		"http://provider.first.java6.webservice.vogella.de/",

		"WiseQuoteServerService"));

				WiseQuoteServer  servicePortWithUrl
= serviceWithUrl

		.getWiseQuoteServerPort();

		System.out.println(servicePortWithUrl.getQuote("fun"));
```

```
            System.out.println(servicePortWithUrl.getQuote("work")
);

                  } catch (MalformedURLException e) {

                        e.printStackTrace();

                  }


            }

}
```

Consuming Web services in Java requires just two steps now. The first step is to create proxy classes. The second step is to go ahead and use them.

4. SOFTWARE REQUIREMENTS SPECIFICATION

Ultimately the requirement phase translates the ideas whatever is in the mind of client (the input) into a formal document (the output of the requirement phase.). In a more general way the SRS is a document that completely describes "What" the proposed system should do without describing "How" the software will do it.

PURPOSE **:** The purpose of the project is to develop a system which is user friendly, easy to use , maintain and satisfies all the requirements of the user.

PERFORMANCE REQUIREMENT

1) The operation time should be small and the throughput should be high.

2) It should produce timely and accurate result.

SOFTWARE QUALITY ATTRIBUTES

i) **Maintainability** – Since it is directly associated with the database, so there is very little maintainability problem with this tool.

ii) **Portability** – Since there is very limited usage of separate forms, this tool is very much portable. This tool uses several canvases on the same form.

iii) **Flexibility** – This tool is very much flexible for future enhancements

FEASIBILITY STUDY

The feasibility study concerns with the consideration made to verify whether the system fit to be developed in all terms. Once an idea to develop software is put forward the question that arises first will pertain to the feasibility aspects.

There are different aspects in the feasibility study:

- Operational Feasibility.
- Technical Feasibility.
- Economical Feasibility.

OPERATIONAL FEASIBILITY:

There in no difficulty in implementing the system, if the user has the knowledge in internal working of the system. Therefore, it is assumed that he will not face any problem in running the system. The main problem faced during development of a new system is getting acceptance from the users. As users are responsible for initiating the development of a new system this is rooted out.

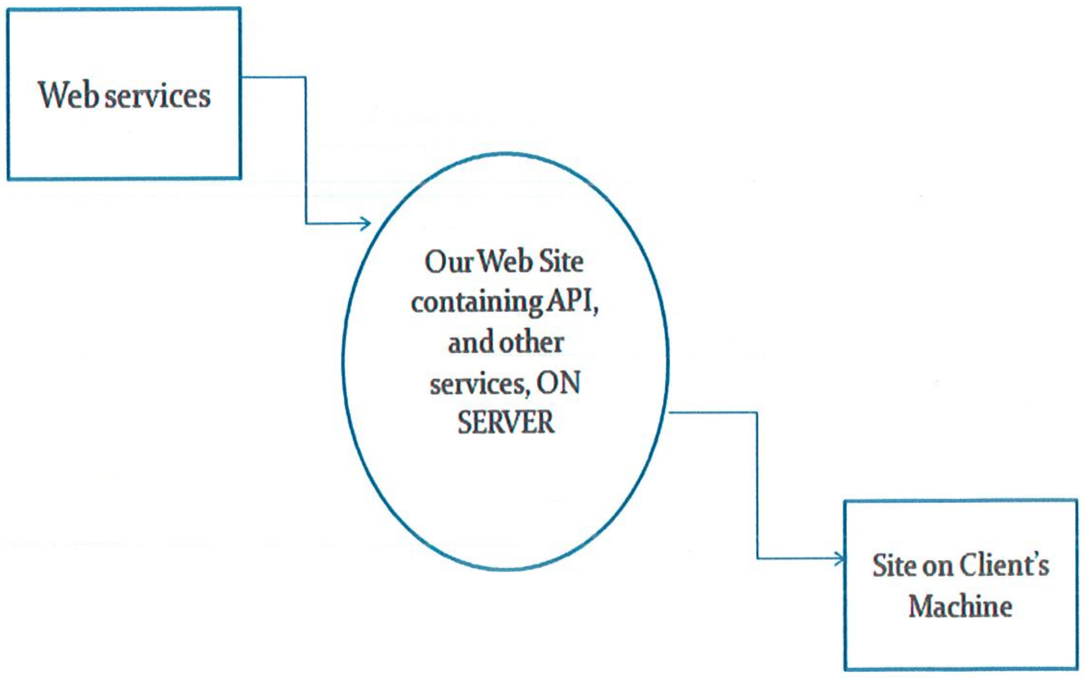
TECHNICAL FEASIBILITY:

Technical feasibility deals with the study of function, performance, and constraints like resources availability, technology, development risk that may affect the ability to achieve an acceptable system.

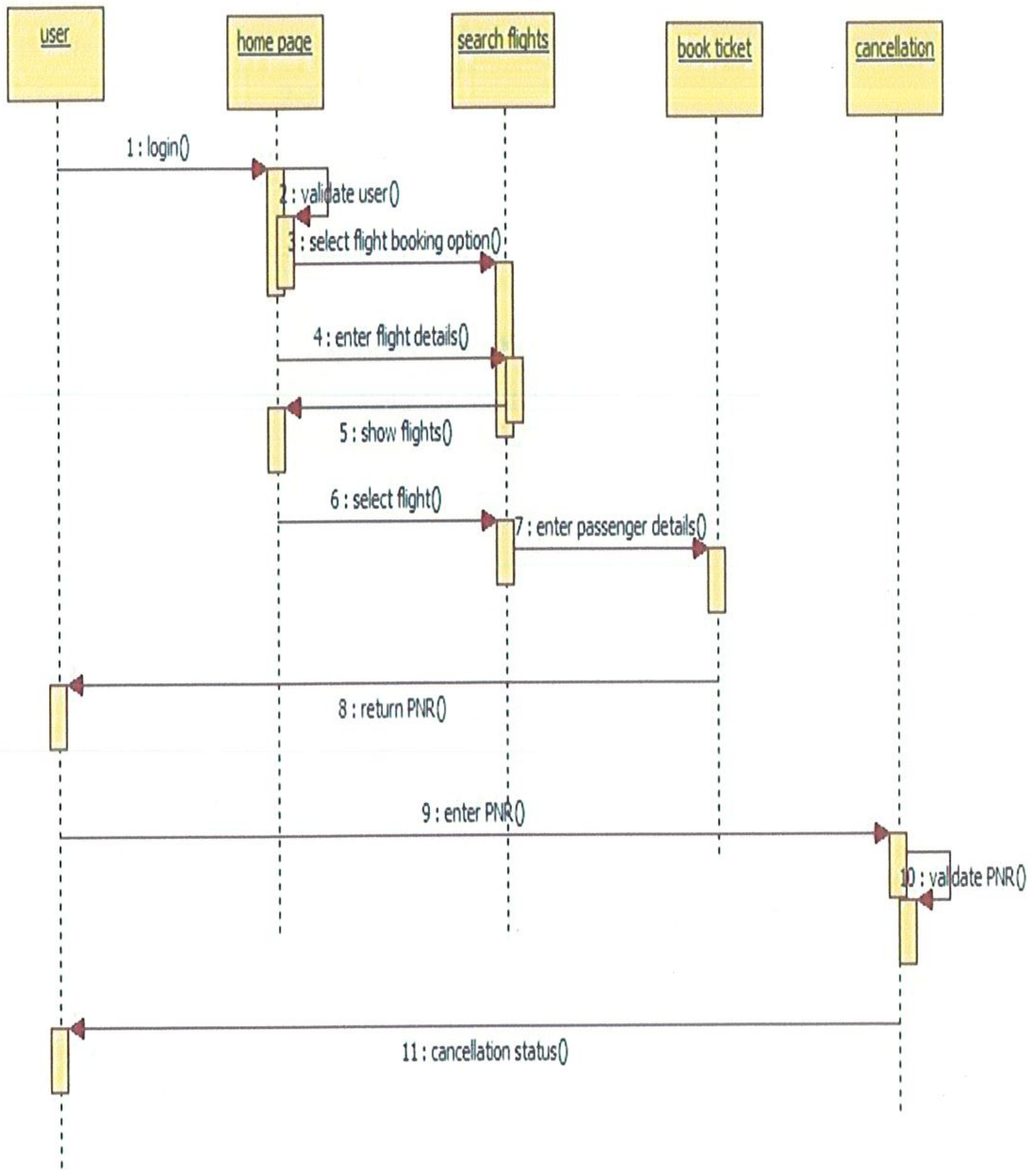
ECONOMICAL FEASIBILITY:

One of the factors, which affect the development of a new system, is the cost it would incur. The existing resources available in the company are sufficient for implementing the proposed and hence no extra cost has to be incurred to run the system developed. Thus, the system is financially feasible.

DATA FLOW DIAGRAM



SEQUENCE DIAGRAM



5. SYSTEM REQUIREMENTS

Operating System:	Windows XP, Vista, Windows 7,
Number of PCs:	2-3
Asp.net:	Microsoft Visual Studio 2008
Server:	Sql server 2005
Hardware:	Laptop/PC with 1 GB RAM, 2 GB free memory for Visual Studio Installation

• REQUIREMENTS ANALYSIS

In the requirements analysis phase:

(a) The problem is specified along with the desired service objectives (goals).

(b) The constraints are identified.

• SPECIFICATION PHASE

In the specification phase the system specification is produced from the detailed definitions of (a) and (b) above. This document should clearly define the product function.

Note that in some text, the requirements analysis and specifications phases are combined and represented as a single phase.

• SYSTEM AND SOFTWARE DESIGN PHASE

In the system and software design phase, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:

a. Data base
b. Software architecture
c. Algorithmic detail and
d. Interface representations

The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'.

• IMPLEMENTATION AND TESTING PHASE

In the implementation and testing phase stage the designs are translated into the software domain:

- ➢ Detailed documentation from the design phase can significantly reduce the coding effort.
- ➢ Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.

• INTEGRATION AND SYSTEM TESTING PHASE

In the integration and system testing phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer [Deliverable – The software product is delivered to the client for acceptance testing.]

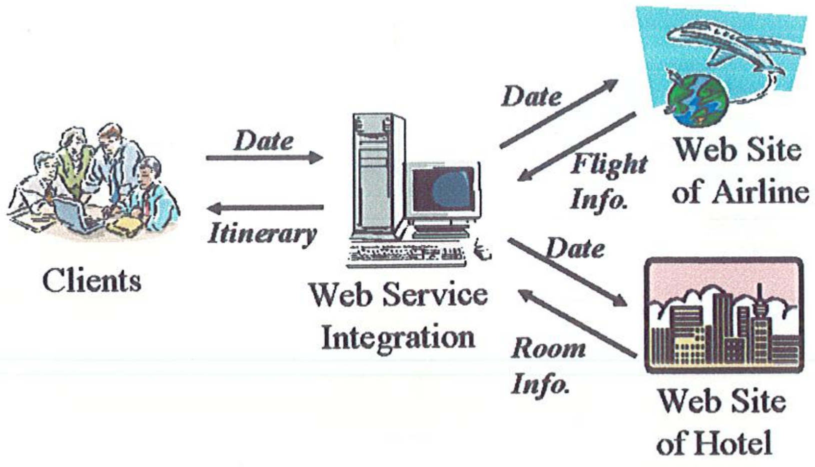
• MAINTENANCE PHASE

The maintenance phase the usually the longest stage of the software. In this phase the software is updated to:

- Meet the changing customer needs
- Adapted to accommodate changes in the external environment
- Correct errors and oversights previously undetected in the testing phases
- Enhancing the efficiency of the software

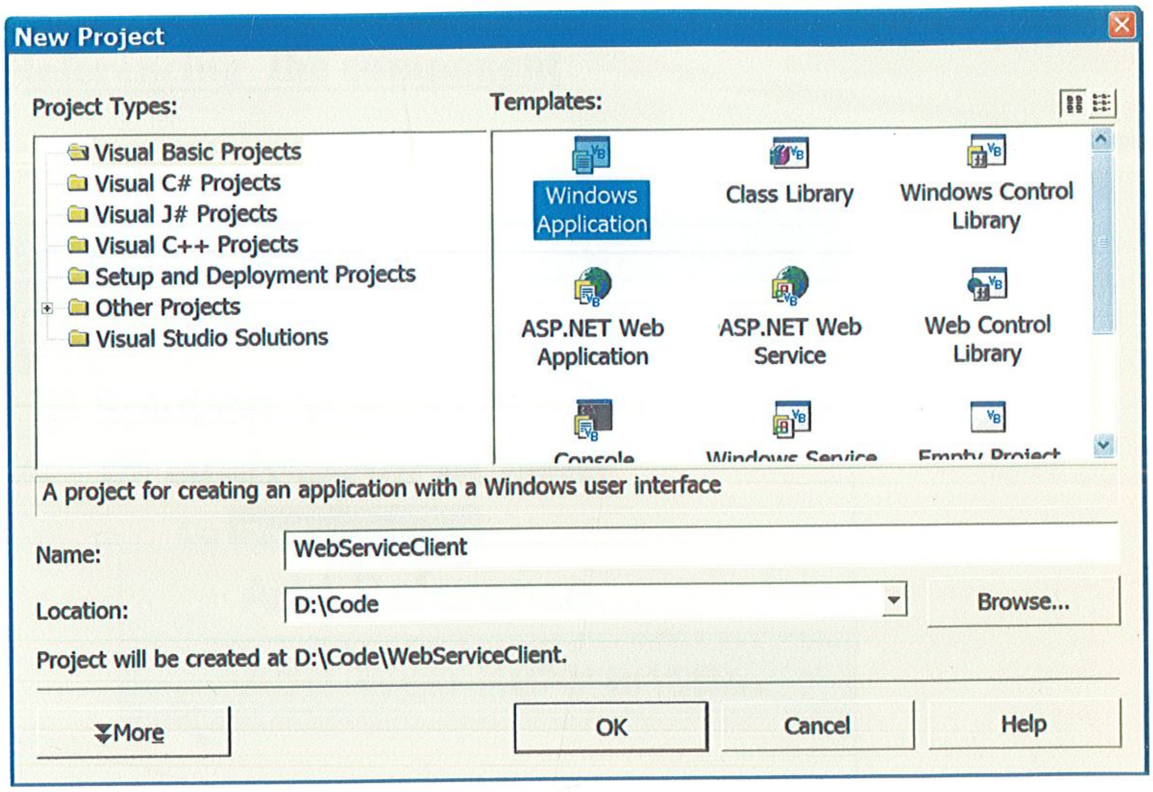
Observe that feed back loops allow for corrections to be incorporated into the model. For example a problem/update in the design phase requires a 'revisit' to the specifications phase. When changes are made at any phase, the relevant documentation should be updated to reflect that change.

6. PROJECT WEBSITE USING .NET



ASP .Net pages are often called web forms because they almost always contain a server side forms element. Controls can be dragged onto the design surface from the toolbox, or created programmatically in code. We kick off with a investigation of the steps involved in working with HTML Controls, as we process a simple form in .NET. We are then introduced to Web Forms and learn how they can be used with HTML Controls and Web Controls to create dynamic Web pages. We'll then move on to explore the processes behind handling page navigation, understand postback, and look at formatting controls with CSS. Finally, armed with this knowledge, we launch into the development of a navigation menu and Web form for our own intranet application.

Building a client



A project for creating an application with a Windows user interface

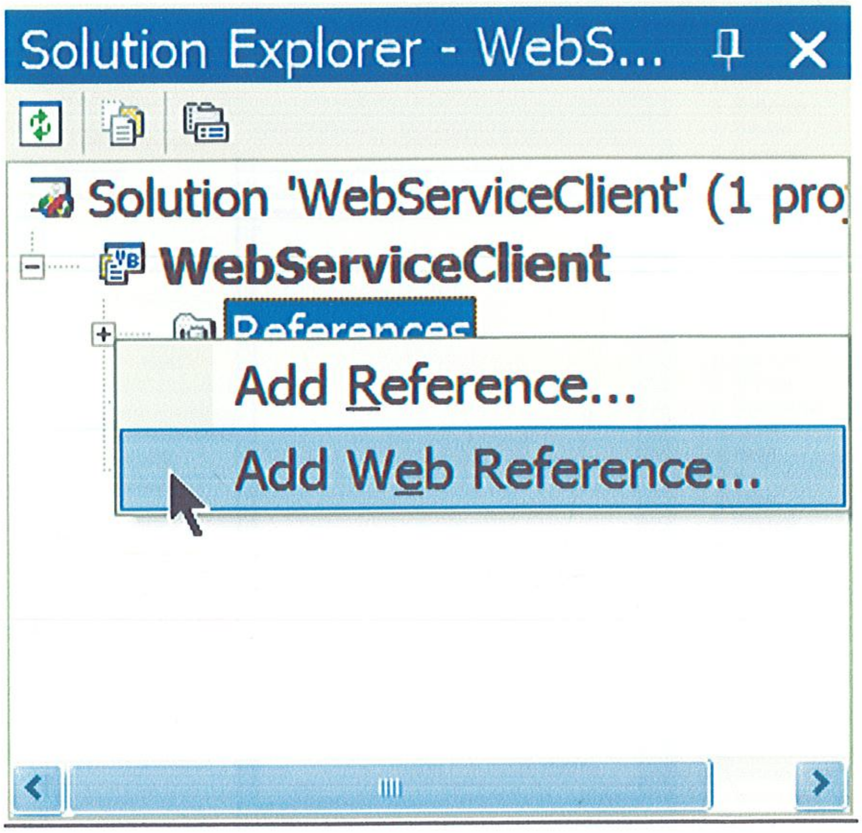
Name: WebServiceClient

Location: D:\Code

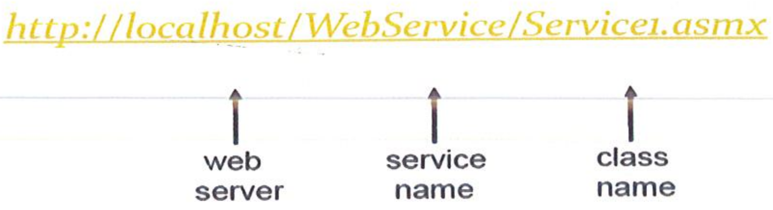
Project will be created at D:\Code\WebServiceClient.

- ASPX pages handled by a handler that facilitate the page lifecycle and events (such as Page_Load, PreRender, and control events).
- Uses ViewState to encoded state-specific information otherwise lost in the stateless nature of HTTP.
- Extensive controls library to abstract functionality. Buttons, textboxes, etc.
- Extensible.Web Forms have a .aspx file extension

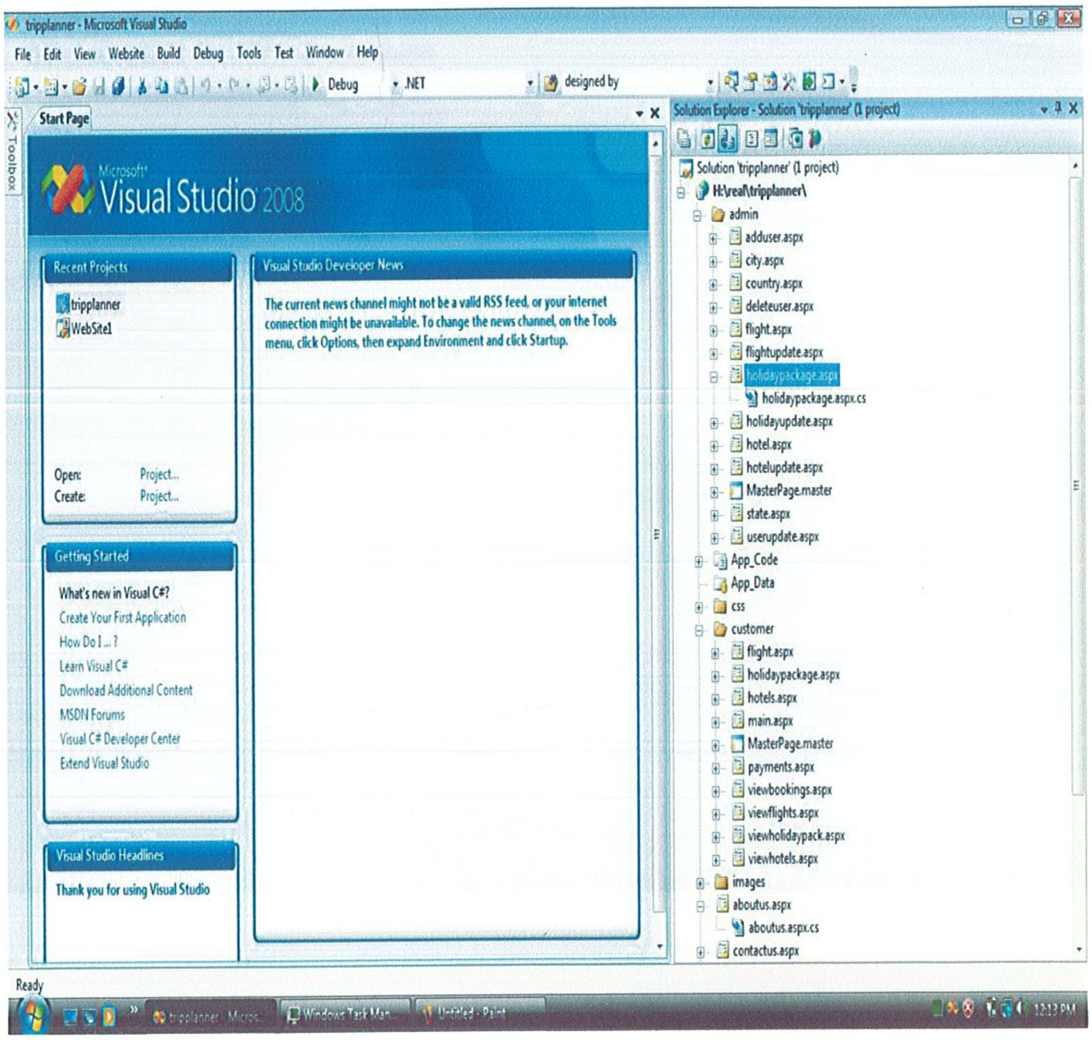
Referencing the component


Solution Explorer - WebS...
Solution 'WebServiceClient' (1 proj
WebServiceClient
References
Add Reference...
Add Web Reference...

We go to project references, right-click, Add webreference, then we type URL for web service, e.g.



PROJECT WEBSITE

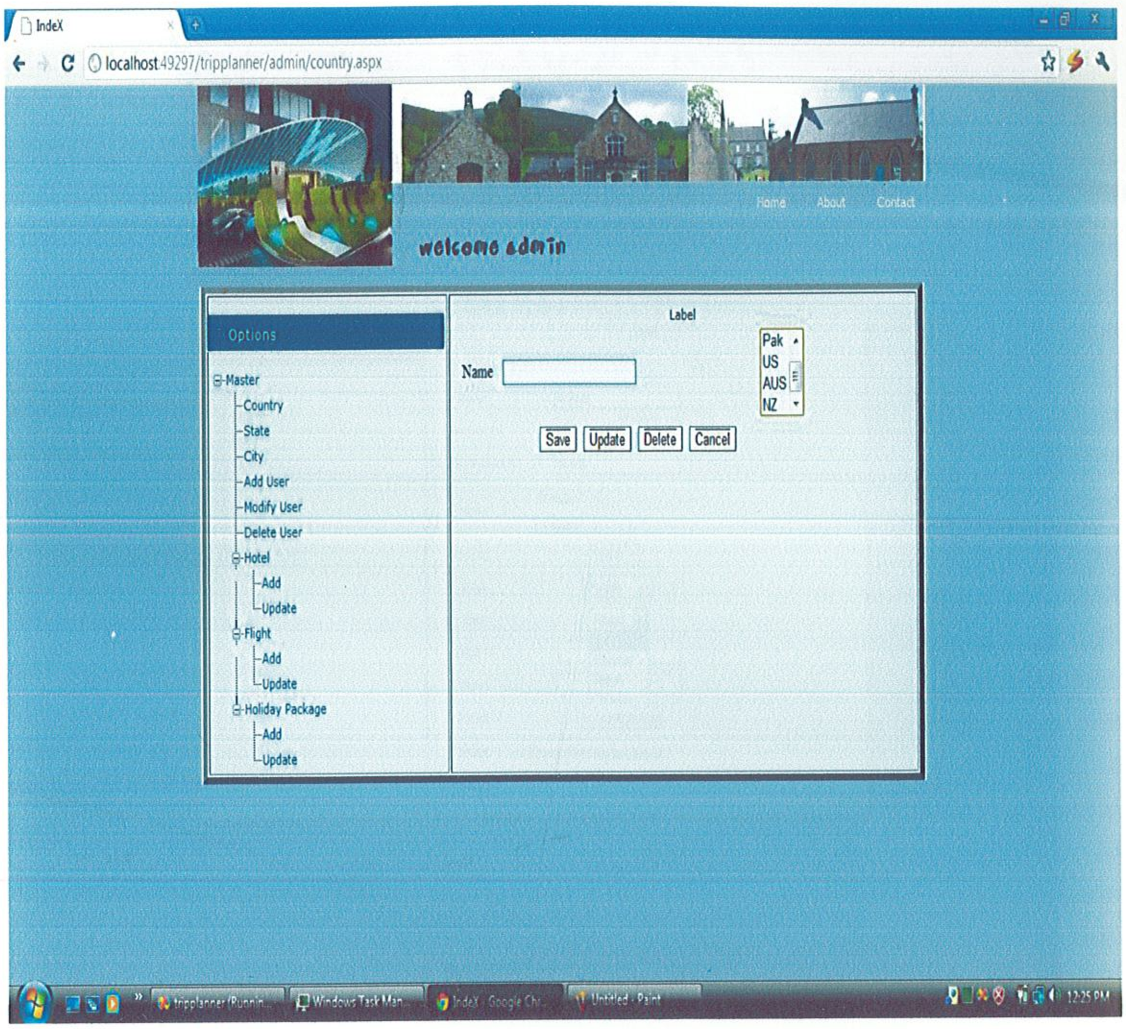


The current news channel might not be a valid RSS feed, or your internet connection might be unavailable. To change the news channel, on the Tools menu, click Options, then expand Environment and click Startup.

Thank you for using Visual Studio

```
Solution 'tripplanner' (1 project)
  H:\real\tripplanner\
    admin
      adduser.aspx
      city.aspx
      country.aspx
      deleteuser.aspx
      flight.aspx
      flightupdate.aspx
      holidaypackage.aspx
        holidaypackage.aspx.cs
      holidayupdate.aspx
      hotel.aspx
      hotelupdate.aspx
      MasterPage.master
      state.aspx
      userupdate.aspx
    App_Code
    App_Data
    css
    customer
      flight.aspx
      holidaypackage.aspx
      hotels.aspx
      main.aspx
      MasterPage.master
      payments.aspx
      viewbookings.aspx
      viewflights.aspx
      viewholidaypack.aspx
      viewhotels.aspx
    images
    aboutus.aspx
      aboutus.aspx.cs
    contactus.aspx
```

Administrator Interface

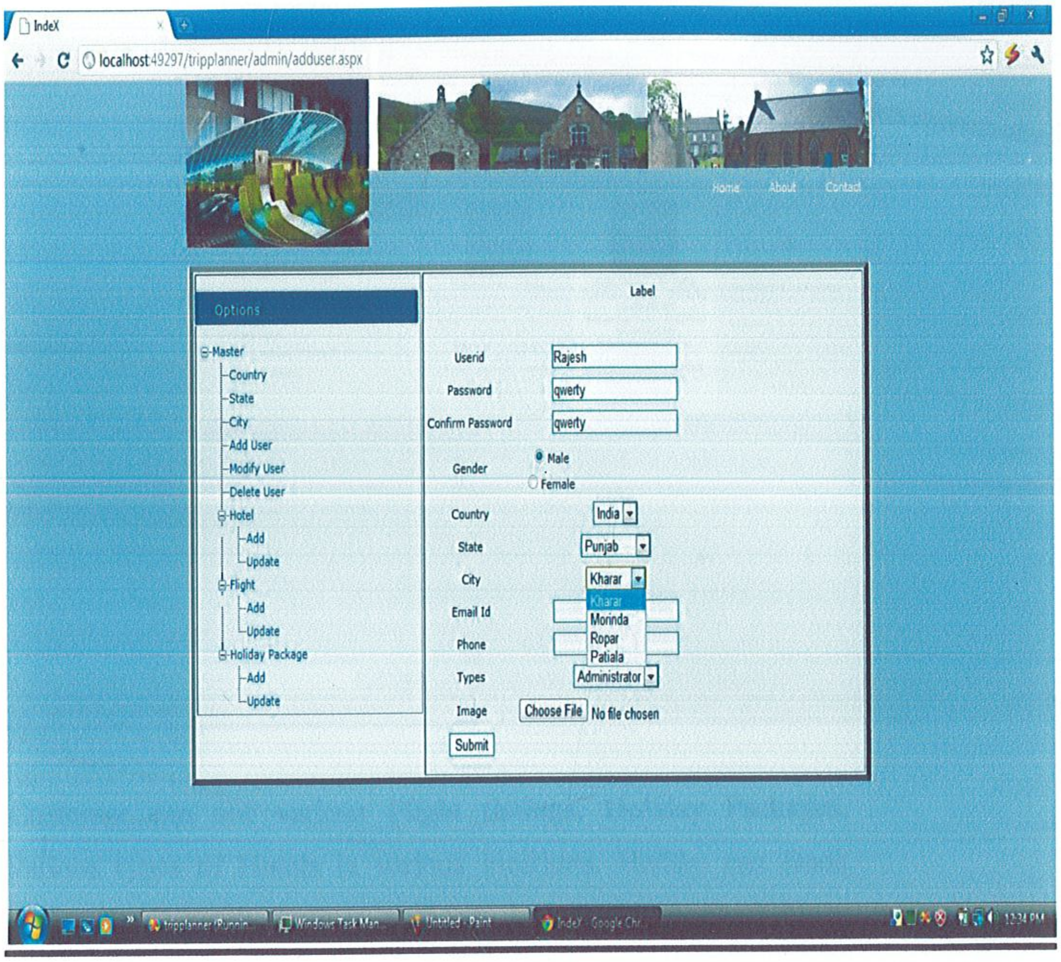


The admin manages the whole database. Admin adds all the information that can be accessed by users later on. Admin adds and modifies the following things to the SQL database.

Add State	Update Hotel
Add City	Modify User
Add User	Delete User
Add Hotel	Update holiday package

Add Flight Update Flight

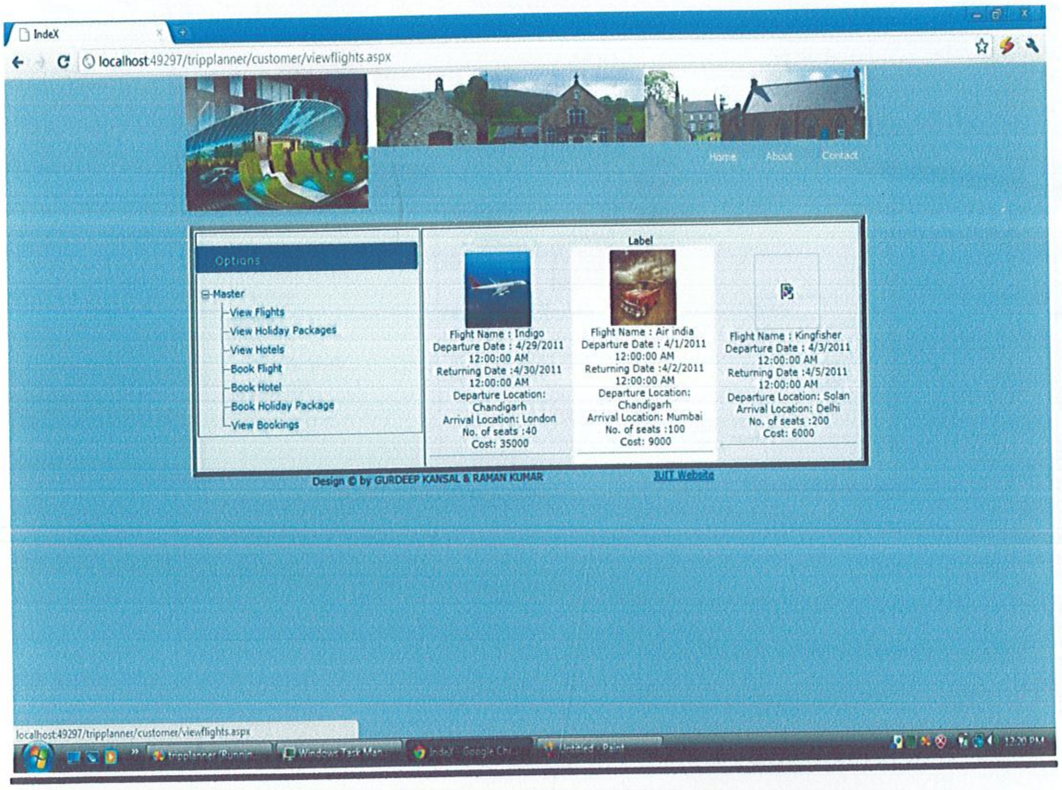
Add Holiday package



Userid	Rajesh
Password	qwerty
Confirm Password	qwerty
Gender	● Male ○ Female
Country	India
State	Punjab
City	Kharar (Kharar, Morinda, Ropar, Patiala)
Email Id	
Phone	
Types	Administrator
Image	No file chosen

This is how an Administrator adds a new user directly from his side. A customer needs a unique user id, password, country name, city name, phone number. UserId is the primary key in the database. The Countries, cities, States that are already added by the administrator are shown while registering a user. The Sql database stores all the values in a table named Customer with user id as its primary key.

CUSTOMER INTERFACE



Flight Name : Indigo
Departure Date : 4/29/2011 12:00:00 AM
Returning Date :4/30/2011 12:00:00 AM
Departure Location: Chandigarh
Arrival Location: London
No. of seats :40
Cost: 35000

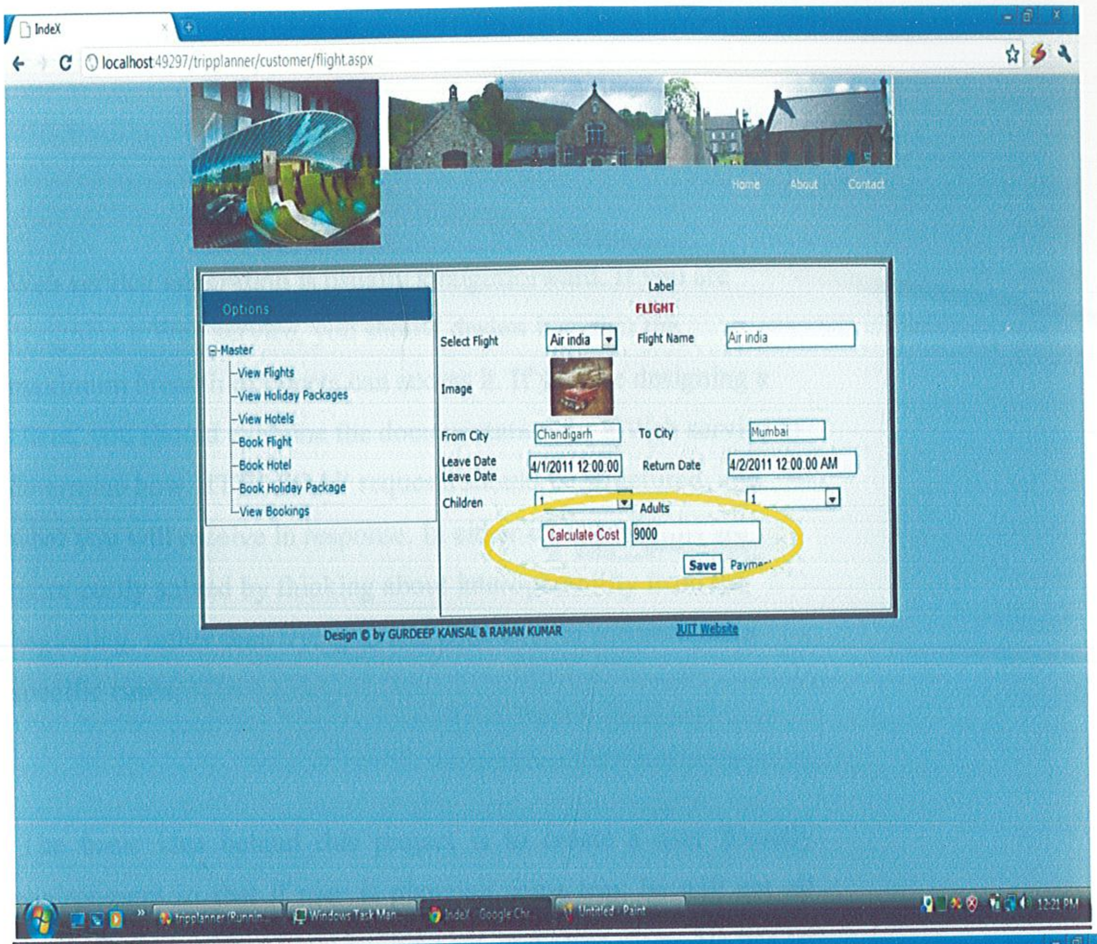
Flight Name : Air india
Departure Date : 4/1/2011 12:00:00 AM
Returning Date :4/2/2011 12:00:00 AM
Departure Location: Chandigarh
Arrival Location: Mumbai
No. of seats :100
Cost: 9000

Flight Name : Kingfisher
Departure Date : 4/3/2011 12:00:00 AM
Returning Date :4/5/2011 12:00:00 AM
Departure Location: Solan
Arrival Location: Delhi
No. of seats :200
Cost: 6000

Design © by GURDEEP KANSAL & RAMAN KUMAR
JUIT Website

Customer can see various Flight timings, Holiday Packages, various types of Hotels in various locations. He/She can Book flights, book hotels, holiday packages and already booked things by Him/Her.

While booking an order the cost is automatically calculated and is shown in the circled area on the web page. The customer can either pay for the booking right away by clicking the "Payment" option from his credit card or He/ She can save the booking order to complete the booking process in case of insufficient funds in the credit card account.


IndeX

localhost:49297/tripplanner/customer/flight.aspx

Home About Contact

Options

Master
- View Flights
- View Holiday Packages
- View Hotels
- Book Flight
- Book Hotel
- Book Holiday Package
- View Bookings

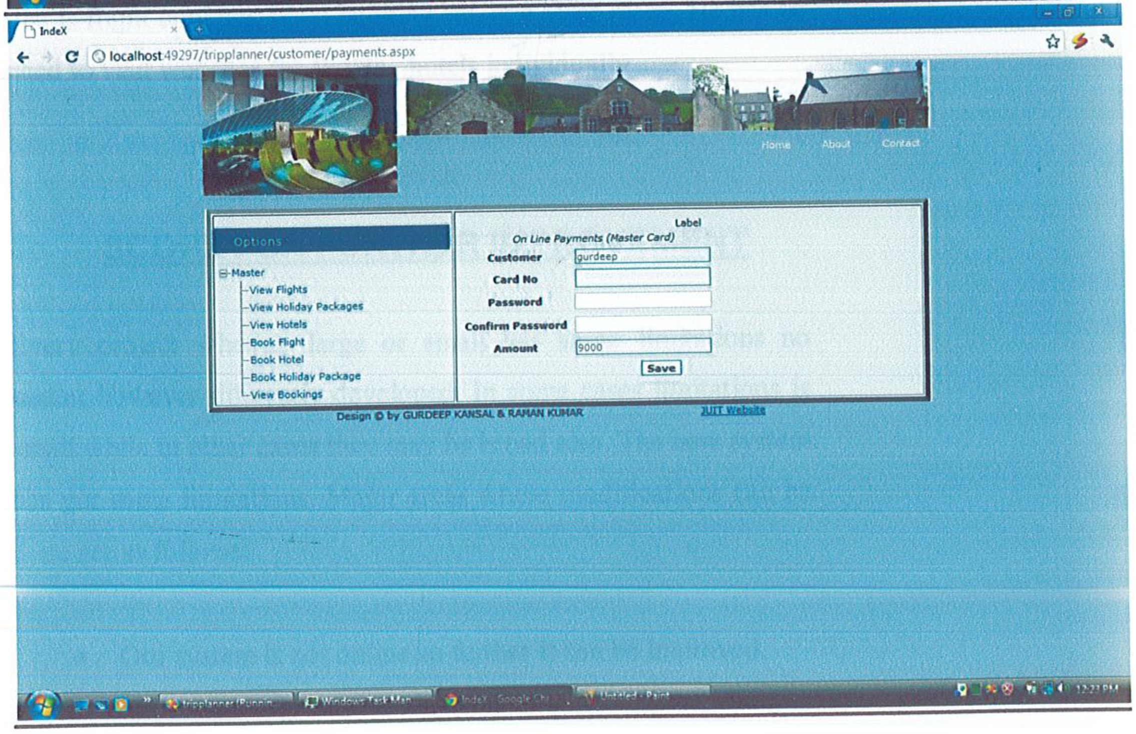
Label

FLIGHT

Select Flight	Air india	Flight Name	Air india
Image			
From City	Chandigarh	To City	Mumbai
Leave Date	4/1/2011 12:00:00	Return Date	4/2/2011 12:00:00 AM
Children	1	Adults	1
Calculate Cost	9000		

Save Payment

Design © by GURDEEP KANSAL & RAMAN KUMAR JUIT Website


IndeX

localhost:49297/tripplanner/customer/payments.aspx

Home About Contact

Options

Master
- View Flights
- View Holiday Packages
- View Hotels
- Book Flight
- Book Hotel
- Book Holiday Package
- View Bookings

Label

On Line Payments (Master Card)

Customer	gurdeep
Card No	
Password	
Confirm Password	
Amount	9000

Save

Design © by GURDEEP KANSAL & RAMAN KUMAR JUIT Website

CONCLUSION

Web service integration is usually straightforward. If you are designing a Web service, you should design it so that the maximum breadth of clients can access it. If you are designing a client, you should examine the documentation for a Web service to determine how HTTP SOAP requests should be structured, and what you will receive in response. In either case, problems are more easily solved by thinking about interoperability from the beginning, rather than trying to add code later to accommodate specific cases.

The basic idea behind this project is to create a user friendly environment so that if user is planning some trip, he will get all that is flight booking, hotel booking etc. at a same place. He has no need to visit websites for flights , hotels individually.

SCOPE FOR FURTHER IMPROVEMENT

Every project whether large or small has some limitations no matter however diligently developed. In some cases limitations is small while in other cases they may be broad also. The new system has got some limitations. Major areas where modifications can be done are as follows:

- Our system is not online so further it can be improved.

- The security is limited so some additional arrangement could be made to provide more security to the system.

- There is no provision of complain handling so further it can be added.

MAIN ACHIEVEMENT OF THE SYSTEM

With the help of this project user can book tickets of different flights, user can book hotels room, search for hotels by entering the specific locations. So user has no need to visit individual sites of different flights. He can get all at a single place. This makes our project more user friendly.
After entering into the project, user is left with several options via
The user can search flights.
The user can view the existing lists of hotels .
The user can of course see rates for hotels and fare for flights as well as other modes of transportation.
The user can also do several administrative works like managing rates, addition or modification of hotels and flights & customer entries.
User can do payment of hotels and flights by using credit cards or debit cards.

It is a computerized system, which can be used very easily. User does not has any need for any programming language.

REFERENCES

1 .Web Services-Markus Mitterer | Mathias Willburger

2. Web Services. Concepts. Architecture and Applications G. Alonso F. Casati, H. Kuno, V. Machiraju Springer Verlag2004

3. Understanding Web Services
http://www.webopedia.com/DidYouKnow/ Computer_
Science/2005/web_services.asp]

4. Tidwell, D. and Snell, J. Programming Web Services with soap.

5. Microsoft VisualStudio.NET http://msdn.microsoft.com/vstudio/

6. The .NET Framework class library:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemXml.asp

7. Amazon, "Web Services,"
http://www.amazon.co.jp/, 2006.

8. OASIS, "Advancing Web Services Discovery Standard,"
http://www.uddi.org/, 2004.

9. The Java™ Web Services Tutorial
Copyright © 2005 Sun Microsystems, Inc., 4150 Network Circle,
Santa Clara, California 95054, U.S.A.

10. **GEOSPATIAL WEB SERVICE INTEGRATION AND MASHUPS FOR WATER RESOURCE APPLICATIONS**

C. Granella, *, L. Díaza, M. Goulda
aCenterfor Interactive Visualization, Department of Information Systems, UniversitatJaumeI, E-12071 Castellón,
Spain -(carlos.granell, laura.diaz, gould)@uji.es

11. **Planning Based Integration of Web Services**

Alfredo Milani, Fabio Rossi, Simonetta Pallottelli
Department of Mathematics & Computer Science
University of Perugia
Via Vanvitelli, 1-06100 Perugia, Italy
milani@dipmat.unipg.it