P-1534

EFFICIENT ACCESS TO WEB SERVICES

KCT LIBRARY Coimbatore-641 006

By
P. KUMARESAN
Register No: 71203405007

Of
Kumaraguru college of Technology,
Coimbatore-641006

A PROJECT REPORT
Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements for the award of the degree

Of
MASTER OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

JUNE 2005

BONAFIDE CERTIFICATE

Certified that this project report titled **Efficient Access to Web Services** is the bonafied work of **P.KUMARESAN** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible] | [illegible]
Internal Guide | Head of the Department

Submitted for the University Examination held on 22/6/05

[illegible] | [illegible] 22/6
Internal Examiner | External Examiner

ABSTRACT

In the expanding internet, users used to be able to efficiently access and share web services. Present query infrastructure or the search tool model are used to search web sites and not web services. The objective of the project is to implement a query infrastructure which is used to search web services that are registered in UDDI (Universal Discovery Description and Integration) registry. This query infrastructure coordinates multiparty business transaction.

The proposal is a novel infrastructure that offers complex and optimized query facilities for web services. In a nutshell, users submit declarative queries that the infrastructure resolves through the combined invocation of different web service operation. The query model facilitates the formulation and submission of queries and their transformation into actual web service operations invocations. The users only need to express the needs for information through simple declarative queries over a well defined interfaces.

ஆய்வுச் சுருக்கம்

இன்றைய இணையத் தொழில் நுட்பத்தில் இணைய தளங்களைத் தேடுவுதற்கான வழி வகைகள் உள்ளன. ஆனால் அவை வெப் சர்வீஸ்சை தேடுவதற்கு பயன்படுத்தவில்லை. இந்த ஆய்வின் நோக்கமே வெப் சர்வீஸ்களை தேடுவதற்கான ஒரு கட்டமைப்பை உருவாக்குவதே ஆகும். இந்தக் கட்டமைப்பு UDDI எனப்படும் பொதுவான ஒரு பதிவகத்தில் பதிவு செய்யப்பட்ட வெப்சர்வீஸ்களை கண்டிறிய பயன்படுகிறது, இந்தக் கட்டமைப்பு பலமுனை வலை வர்த்தக பரிமாற்றங்களை ஒருங்கிணைக்கிறது.

இந்தக் கட்டமைப்பு சிக்கலான கேள்விகளையும் ஏற்றுக் கொள்ளக் கூடியது. உபயோகிப்பாளர்கள் கொடுக்கும் கேள்விகளுக்கு வெவ்வேறு வெப் சர்வீஸ்களை சேர்த்துக் தேடுவதன் மூலம் தேடலுக்கான விடை அறியப்படுகிறது. உபயோகிப்பாளர்கள் தாங்கள் விரும்பும் தகவல்களை சின்னஞ்சிறு கேள்விகளை வரையறுக்கப்பட்ட இணைப்புக் குறியீடுகள் மூலம் இணைந்து கேட்டுப் பெறலாம்.

ACKNOWLEDGEMENT

I am extremely thankful to **Dr.K.K.Padmanabhan B.Sc. (Eng.), MTech, Ph.D.,** Principal, Kumaraguru College of Technology who has given me his valuable guidance and an efficient infrastructure that aided me during the course of my education.

With profound sense of gratitude and regards, I acknowledge with great pleasure the guidance and support extended by **Prof.S.Thagasamy BE.(Hons), Ph.D.,** head of the Department of Computer science and Engineering for his encouragement and moral support throughout the course of the project.

I would like to express my indebted gratitude and whole hearted thanks to my guide **Mr.R.Dinesh MS,** Assistant Professor, Department of Computer science and Engineering, Kumaraguru College of technology, for Constant Encouragement throughout the course of work. I also like to express my special thanks to all the staffs and lab technicians in the department of Computer science and Engineering who helped me for the successful completion of the project.

Finally and most importantly, I would like to thank my **beloved parents, friends and other staff members** for giving me endless encouragement and necessary help, during the course of my project.

CONTENT

CHAPTER I

INTRODUCTION

Web services provides a way to describe their interface in enough details to allow a user to build a client application to talk to them. Web services are registered so that potential users can find them easily. This Is done with UDDI. UDDI is the yellow page of web services.

The difficulty in inter enterprise computing today is the problem caused by difference in platforms and transports, which necessitate tedious conversion of data and functionality in order for two business with different architecture to inter operate. In early days, CORBA is used for inter enterprise computing. The disadvantage is it's lack of flexibility and its inability to handle asynchronous messaging in a straight forward manner.

The emergence of web services introduces a new paradigm for enabling the exchange of information across the internet based on open internet standards and technologies using industry standards web services encapsulates application and publish them as services. This services deliver XML based data on the wire and expose it for use on the internet, which can be dynamically located, subscribe, and accessed using a wide range of computing platforms, handheld devices, application and so on. Due to the flexibility of using open standards and protocols and also facilities enterprise application integration, B2B integration and A2A communication across the internet and corporate intranet. In organization heterogeneous application and distributed application architecture, the introduction of web services standardizes the communication and enables inter operability of application based on different programming languages residing on different platforms.

1.1 Web Service

To date, most of the traffic on the Internet has been human-to-human interaction (email) and human-to-server interaction (web pages). In the future, it is expected that a significant portion of traffic will be machine-to-machine interaction – where one program calls another over the Internet and gets back the results that it needs. The language of the request and the response is increasingly becoming XML. This process, of calling functionality over the network (which could be over the public Internet, or across different parts of a private, corporate network) and the emerging XML-based standards for accomplishing this communication are referred to collectively as Web Services.

Web services are the fundamental building blocks in the move to distributed computing on the internet. Web services are becoming the platform for application integration. XML web services provide a way to describe their interface in enough details to allow a user to build a client application to talk to them. This description is usually provided in an XML document called a web service descriptive language(WSDL) document. We define an XML web services as a software service exposed on the web through SOAP, described with a WSDL file and registered in UDDI.

Web service are based on the concept of service oriented architecture. SOA is the latest evolution of distributed computing which enables software components including application functions, object and process from different systems, to be exposed as service. According to Gartner research "Web services loosely coupled software components delivered over internet standard technologies".

In short, web services are self describing and modular business application that exposes the business logic as services over the internet through programmable interfaces and using internet protocols for the purpose of providing ways to find, subscribe and invoke those services.

Based on XML standard web services can be developed as loosely coupled application components using any programming languages, any protocols, or any platform. This facilitates delivery business application as a service accessible to anyone, anytime, at any location and using any platform.

Web services enable business to communicate, collaborate and conduct business transaction using light weight infra structure. Web services are based on XML messaging which means the data exchanged between the web service provider and the user are defined in XML. Web service provide a cross platform integration of business application over the internet.
A web service exhibits the following characteristics:

1. A web service is a web resource. Web services can be accessed using platform independent and language neutral web protocols, such as HTTP. These web protocols ensure easy integration of heterogeneous environment.
2. A web service provides an interface – a web API that can be called from another program. This application to application programming interface can be invoked from any type of application. The web API provides access to the application logic that implements the service.
3. Web services supports loosely coupled connection between systems. Web services communicate by passing XML messages to each other via a web API, which adds a layer of abstraction to the environment that makes the connections flexible and adaptable.

The following are the major technical reasons for choosing web services over web applications

1. Web services can be invoked through XML based RPC mechanism across firewall
2. Web services provide a cross platform, cross language solution based on XML messaging

3. Web services facilitates ease of application integration using a light weight infrastructure without affecting scalability
4. Web services enable interoperability, heterogeneous application

Web Services components

Several essential activities need to happen in any service-oriented environment: A Web service needs to be created, and its interfaces and invocation methods must be defined.

1. A Web service needs to be published to one or more intranet or Internet repositories for potential users to locate.
2. A Web service needs to be located to be invoked by potential users.
3. A Web service needs to be invoked to be of any benefit.
4. A Web service may need to be unpublished when it is no longer available or needed.
5. A Web Services architecture then requires three fundamental operations: publish,find, and bind. Service providers publish services to a service broker. Service requesters find required services using a service broker and bind to them.

Web Services Architecture

Typical web services architecture models consists of three logical components as core building blocks mapping the operational roles and relationships of a web services environment.

Service Container acts as the web services run time environment and hosts the service provider. It defines the service deployment and service

administration in addition it also handles the registration of the service description in the service registries.

Service registry : The services registry hosts the published services and acts as the broker providing the facility to publish and store the description of web services registered by the services provider in addition, it defines a common access mechanisms for the service requestors for location the registered services.

Services Delivery acts as the web services client run time environment by looking up the services registry to fine the required services and invoking them from the service providers.

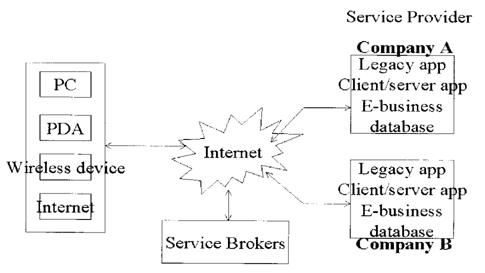


Figure 1.1 – Web Service Architecture

Here is the list of three fundamental technologies on the web service world
SOAP – Simple Object Access Protocol
WSDL – Web Service Descriptive Language
UDDI – Universal Description, Discovery and Integration

1.2 SOAP – Simple Object Access Protocol

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.

SOAP consists of three parts:

- The SOAP envelope construct defines an overall framework for expressing **what** is in a message; **who** should deal with it, and **whether** it is optional or mandatory.

- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined datatypes.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity. In addition to the SOAP envelope, the SOAP encoding rules and the SOAP RPC conventions, this specification defines two protocol bindings that describe how a SOAP message can be carried in HTTP messages either with or without the HTTP Extension Framework.

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but SOAP messages are often combined to implement patterns such as request/response. All SOAP messages are encoded using XML. A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. This XML document is referred to as a SOAP message. A SOAP message contains the following:

- The Envelope is the top element of the XML document representing the message.
- The Header is a generic mechanism for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. SOAP defines a few attributes that can be used to indicate who should deal with a feature and whether it is optional or mandatory
- The Body is a container for mandatory information intended for the ultimate recipient of the message. SOAP defines one element for the body, which is the Fault element used for reporting errors.

Using SOAP in HTTP

Binding SOAP to HTTP provides the advantage of being able to use the formalism and decentralized flexibility of SOAP with the rich feature set of HTTP. Carrying SOAP in HTTP does not mean that SOAP overrides existing semantics of HTTP but rather that the semantics of SOAP over HTTP maps naturally to HTTP semantics. SOAP naturally follows the HTTP request/response message model providing SOAP request parameters in a HTTP request and SOAP response parameters in a HTTP response. Note, however, that SOAP intermediaries are NOT the same as HTTP intermediaries. That is, an HTTP intermediary addressed with the HTTP Connection header field cannot be expected to inspect or process the SOAP entity body carried in the HTTP request.

1.3 WSDL – Web Service Descriptive Language

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, like SOAP 1.1, HTTP GET/POST, and MIME.

As communications protocols and message formats are standardized in the web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication end points capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication.

A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** which are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitutes a reusable **binding**. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **Types**– a container for data type definitions using some type system.
- **Message**– an abstract, typed definition of the data being communicated.
- **Operation**– an abstract description of an action supported by the service.
- **Port Type**–an abstract set of operations supported by one or more endpoints.
- **Binding**– a concrete protocol and data format specification for a particular port type.
- **Port**– a single endpoint defined as a combination of a binding and a network address.
- **Service**– a collection of related endpoints.

WSDL does not introduce a new type definition language. WSDL recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification (XSD) as its canonical type system. However, since it is unreasonable to expect a single type system grammar to be used to describe all message formats present and future, WSDL allows using other type definition languages via extensibility.

In addition, WSDL defines a common **binding** mechanism. This is used to attach a specific protocol or data format or structure to an abstract message, operation, or endpoint. It allows the reuse of abstract definitions.

KCT LIBRARY Coimbatore-641 006

1.4 UDDI

The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web Services. The term "Web Service" describes specific business functionality exposed by a company, usually through an internet connection, for the purpose of providing a way for another company or software program to use the service.

Web services are becoming back bone for electronic commerce. When we want to find out which business partner have which business service, the ability to discover the answer can become difficult. One option is to call every partner on the phone and try to find which service they have.

Another way to solve this problem is through an approach that uses a web services description file on each company web site. Web crawlers work by accessing a registered URL and are able to discover and index text found on nests of web pages. This approach is again dependent on the ability of a crawler to locate each web site and the location of the service description file on that web site. This distributed approach is potentially scalable but lacks a mechanism to insure consistency in service description formats and for the easy tracking of changes as they occur.

UDDI takes an approach that relies upon a distributed registry of business and their service descriptions implemented in a common XML format.

The core component of the UDDI project is the UDDI business registration, an XML file used to describe a business entity and its web services. Conceptually, the information provided in a UDDI business registration consists of three components: "White page" includes address, contact and known identifiers; "Yellow page" include industrial categorizations based in standard taxonomies;

and "Green page", the technical information about the services that are exposed by the business. Green page include references to specifications for web services, as well as support for pointers to various file and URL based discovery mechanisms if required.

Using UDDI

UDDI includes the shared operation of a business registry on the web. For the most part, programs and programmers use the UDDI Business Registry to locate information about services and, in the case of programmers, to prepare systems that are compatible with advertised web services or to describe their own web services for others to call. The UDDI Business Registry can be used at a business level to check whether a given partner has particular web service interface, to find companies in a given industry with a given type of service, and to locate information about how a partner or intended partner has exposed a web service in order to learn the technical details required to interact with that service.

UDDI is the yellow page of web service. A UDDI directory entry is an XML file that describes a business and the services it offers. There are three parts to an entry in the UDDI directory. The "White page" describes the company offering the service, name, address etc. the "Yellow page" includes industrial category, the "Green page" describes the interface to the service in enough details for someone to write an application to use the web services.

The Universal Description, Discovery and Integration (UDDI) specification describes a conceptual cloud of Web services and a programmatic interface that define a simple framework for describing any kind of Web service. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services. Using the UDDI discovery services, businesses individually

register information about the Web services that they expose for use by other businesses. This information can be added to the UDDI business registry either via a Web site or by using tools that make use of the programmatic service interfaces described in the UDDI Programmer's API Specification. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other on a regular basis. Once a business registers with a single instance of the business registry service, the data is automatically shared with other UDDI root nodes and becomes freely available to anyone who needs to discover what Web services are exposed by a given business.

1.5 XML

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined. You must define your own tags
- XML uses a Document Type Definition (DTD) or an XML Schema to describe the data
- XML with a DTD or XML Schema is designed to be self-descriptive
- XML is a W3C Recommendation

The main difference between XML and HTML

XML was designed to carry data. XML is not a replacement for HTML. XML and HTML were designed with different goals. XML was designed to describe data and to focus on what data is. HTML was designed to display data and to focus on how data looks. HTML is about displaying information, while XML is about describing information.

XML was not designed to DO anything. Maybe it is a little hard to understand, but XML does not DO anything. XML was created to structure, store and to send information. The following example is a note to Student from Teacher, stored as XML:

```
<note>
<to>Student</to>
<from>Teacher</from>
<heading>Reminder</heading>
<body>Don't forget to prepare for exam this weekend!</body></note>
```

The note has a header and a message body. It also has sender and receiver information. But still, this XML document does not DO anything. It is just pure information wrapped in XML tags. Someone must write a piece of software to send, receive or display it.

XML is free and extensible. XML tags are not predefined. You must "invent" your own tags. The tags used to mark up HTML documents and the structure of HTML documents are predefined. The author of HTML documents can only use tags that are defined in the HTML standard (like <p>, <h1>, etc.). XML allows the author to define his own tags and his own document structure.

XML is a complement to HTML. XML is not a replacement for HTML. It is important to understand that XML is not a replacement for HTML. In future Web development it is most likely that XML will be used to describe the data, while HTML will be used to format and display the same data. XML is a cross-platform, software and hardware independent tool for transmitting information.

XML is going to be everywhere.

CHAPTER II

LITERATURE SURVEY

1. Coordinating Business Transaction on the Web - *Sanjay Dalal and Sazi Temel, IEEE Internet Computing*

The shortcomings of traditional transaction have become apparent as Web Services have evolved for integrating inter-enterprise processes and applications. To guarantee consistent outcomes and correct execution, most business to business collaborative applications require transactioal support. These applications often involve long running computations, loosely coupled systems, and components that do not share data, location or administrations.

2. Service Discovery 101 – *Steve Vinoski, IEEE Internet Computing*

A key aspect of the service oriented architecture approach for middleware is that services advertise themselves using directory or lookup services so that prospective clients can find them. The service location and discovery abstractions required to support this aren't much different from those we use to conduct business with other people. As a result, we can gain insights into how distributed service discovery systems works by comparing them to everyday human oriented service discovery approaches.

3. Infrastructure for E-Government Web Services – *Brahim, Rezgui and Ouzzani, IEEE Internet Computing*

The ultimate goal is to improve government citizen interaction through an infrastructure built around the life experience of citizens. The current process for collecting social benefits is time consuming and frustrating, as citizen must

often visit multiple offices located within and outside their hometown. Case officers must use myriad applications, manually determining which will satisfy citizen's individual needs, deciding how to access each form, and combining the results returned by different applications – all without divulging citizen's information to unauthorized entities. To facilitate the use of welfare application, authors wrapped these application in Web services. Adopting Web service in this project, provides value added service. The core of the research is to develop techniques to efficiently access e-government services.

4. Searching the Internet – *Robert and Sangam Pant, IEEE Internet Computing*

Software development would continue to push the frontier of the feasible—interfaces would get more immersive, databases bigger, simulations more fine grained. The surprise, from a technologist's point of view, has been the social phenomena of the World Wide Web. A decade ago, the general population was computer phobic and took their entertainment passively. Now we see advertisements trumpeting HTTP addresses and grandmothers surfing the Net. Meanwhile, everyone and their cousin has turned into a publisher of original HTML material. The networks and devices are going to continue to get faster and cheaper. Our current concerns with network protocols and algorithms for, say, real-time multimedia presentations will seem in 30 years as quaint as tape-sorting algorithms do today. What will continue to be AI-hard is the problem of making sense of the swelling mass of data and misinformation that fills the Web.

That's the problem of searching the Web.

5. Efficient Access to Web service – *Ouzzani and Bouguettaya, IEEE Internet Computing*

For Web services to expand across the Internet, users need to be able to efficiently access and share Web services. The authors present a query infrastructure that treats Web services as first-class objects. It evaluates queries through the invocations of different Web service operations. Because efficiency plays a central role in such evaluations, the authors propose a query optimization model based on aggregating the quality of Web service parameters of different Web services. The model adjusts quality of web services through a dynamic rating scheme and multilevel matching in which the rating provides an assessment of Web services' behavior. Multilevel matching allows the expansion of the solution space by enabling similar and partial answers.

CHAPTER III

EFFICIENT ACCESS TO WEB SERVICES

Using web services consist generally of invoking operations by sending and receiving messages. There are some complex applications, such as a travel package that access diverse web services, needs an integrated and efficient way to manipulate and deliver web services functionality. This project propose a novel infrastructure that offers complex and optimized query facilities for web services. In short, users submit declarative queries that the infrastructure resolves through the combined invocation of different web service operation.

3.1 Example Scenario Of The Problem: Planning A Trip

An example in the context of web services is a tourist who is looking for a good travel plan. The user would start searching for airline sites using a web search engine which would return numerous answers. After selecting a few airline web sites and querying them, user might select one that seemed to offer the best deal. This process continues for the other finding for Hotel reservation and Car rental. This scenario highlights that lay users have to go through to conduct complex queries on the web. The user need a query system where he need only to express his needs for information through simple declarative queries over a well defined interface.

3.2 Traditional Search Techniques

The Web Site searching

There are basically two types of search engine technologies, they are 1. Crawler based search engines and 2. Human powered search engine.

A human powered directory such as Open Directory, depends on humans for its listings. We submit a short description to the directory for the entire web site, or editors write one for sites they review. A search looks for matches only in the descriptions submitted. Changing the web page has no effect on the listing.

Crawler based search engines such as google, create their listing automatically. They crawl or spider the web, then people search through what they have found. If there is any change in web pages, crawler based search engine eventually find these changes and that can affect how the page is listed.

Now a days, both types of search are used for listing by combining two techniques called Hybrid search engines which favor one type of listing over another. An example for hybrid search is MSN Search which present human powered listings from Looksmart and also present crawler based results.

Working of Search engine

The working of a search engine. The user provides keywords and the search service looks for those keywords in the meta-data of web sites that it has listed in its database. Search engines may also provide advanced features like searching within search results, excluding records that contain certain keywords, etc. No matter how advanced the search is, though, it will always depend on keywords.

Future search engines (*smart web crawlers*) will be better than this. They will not only search for keywords, but also find the capabilities of web sites. What if you want to find hotels in a city that fall within a particular range of per night charge? A conventional search engine can get a list of URLs for you to explore and visit each of those hotel sites to check the costs. With smart web crawlers, you can directly look for hotel Web Services that expose their functionality for giving automated quotations and reservations. UDDI enabled smart web crawlers will use HTTP, SOAP, and WSDL to get a list of hotels, and check which of those hotels publish their tariff and allow automatic reservations. You would then invoke the services of those sites to check their charges, and possibly make reservations

3.3 Web Service Search Using UDDI Registry As A Smart Web Crawler

Web crawler works for both ordinary internet users as well as for web service enabled electronic commerce application. If an ordinary Internet user surfs the web using a browser program, smart web crawlers have value-added services to offer the user. They allow user to surf through categorizations and classifications, finding Web Services that belong to a particular industry and explore their interfaces to check whether they offer services of interest to the user. In order for UDDI operators to serve all these purposes, they need very user-friendly and smart graphical user interfaces that allow ordinary Internet users to take full advantage of this new technology.

For a Business to Business Web Service owner running an on-line tourist information and e-commerce show, where finding hotels is part of the business process, using conventional web search methods, owner will have to manually find hotels, talk to them, and design a content management solution that suits both the partner's and their own database systems. Imagine the owner have few hundred hotels as partners, each of them having their own back-end data format. In this case, owner would need to implement coordination and electronic data exchange mechanisms with each individual partner hotel. With smart web

crawlers, the entire operation becomes smooth and automated as UDDI allows the owner to discover Web Services. This, together with WSDL allows Internet users and B2B owners to meaningfully interact with the services that their partners offer. Therefore, whenever a new hotel enters the UDDI registry that user work with, it will automatically start appearing on the tourist B2B site, independent of their web site's application architecture.

Web Services Querying

Web services primary use has so far consisted of invoking operations by sending and receiving messages. Although this meets the requirements of simple applications, complex applications that access diverse Web services need an integrated way to manipulate and deliver Web services' functionalities. In addition, as more companies start to deploy Web services, many will compete by offering similar functionalities — different airlines offering the same connection between two cities, for example. However, the Web services will differ in the way that they offer these functionalities and the conditions for using them. In addition, satisfying users' requests might not require returning exact answers. Indeed, users might be satisfied by alternative or "partial" answers.

As a major step in addressing these challenges, the project propose a new approach to querying Web services and the information flow during the invocation of their operations. Depending on the query, its resolution might lead to the invocation of various Web service operations and the combination of their output results. Our approach emphasizes the optimization of the query-resolution process while allowing partial answers.

CHAPTER IV

QUERY INFRASTRUCTURE MODELING

4.1 QUERY MODEL FOR WEB SERVICES.

To facilitate the interactions between users and web services, we need to represent the space of web services within a given application domain in a generic way. We define a set of domain specific operation called virtual operations, so called because they don't belong to any actual web service for use within a three level query model:

The Query level consists of set of relations that give users an interface for submitting declarative queries to Web services. Different mapping rules define different sets of relations over the virtual operations.

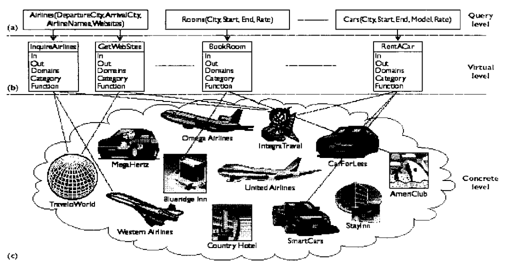


Figure 4.1 - Level of Query Model

The Virtual level consists of the web service like operations a particular application domain typically offers. Example finding hotel room and checking flight availability.

The Concrete level represents the actual web space and services offered by the web.

4.2 Steps to Search through UDDI Registry

1. **Find the businesses of interest**. This forms the logical start of search operations. An Internet surfer will provide search criteria (such as products, product categories, key words, etc.) to begin a search. The intention of including this method is to allow preliminary searching, where further more precise searches will be carried out in subsequent steps. Therefore this method returns a list of summarized results.

2. The next logical step is to find the categories of services that a particular business offers (such as hotel reservation services that a travel and tourism B2B site may offer). Therefore, UDDI provides a method for finding an abbreviated list of services that any particular business offers. Smart web crawlers will use this method to find a list of all services, once the user has selected the business of their choice.

3. **Find all methods of interest**. The user will, in this case, provide search criteria (say, business categorization, which is a very important search criterion), in response to which this method will return a list of methods fulfilling those criteria. Users can search for specific criteria using this method. UDDI, while providing a mechanism for discovery of Web Services, at the same time also provides a very rich and extensible framework for business categorization. The UDDI specification allows development of specialized search services for specific industries on top of general registries.

4. **Find bindings of interest** within a particular business service. When we invoke this method, the UDDI registry will respond by sending a list of all binding templates that this business service contains. Recall from earlier that binding template provides an access point into a Web Service. A binding template may refer to a WSDL interface

Any search through a UDDI registry will be a combination of these basic search methods

4.3 From Relations to Virtual Operations

Relations at the query level define a specific view of the application domain. We represent them as conjunctive queries over virtual operations, thus presenting the virtual operations just as if they were relations. More precisely, let R be the set of relations the query level defines and let VOP be the set of virtual operations. For any relation *Ri* Є R,

$R_i(x_1,x_2,\ldots,x_n)$:- $\Lambda_j\ Vop_j(yj_1,yj_2,\ldots,yj_m)$

where x_i are the attributes of R*i*, *Vopj* Є VOP, and
y_{ji} are the corresponding operation's input and output variables.

This definition means that to get *Ri*'s tuples, we need to invoke the different operations *Vopj*. It does not mandate any order or limit concurrency among these operations. An example of a mapping rule is

Airlines(DepartureCity, ArrivalCity, AirlineNames, WebSites) :-
InquireAirlines(DepartureCity, ArrivalCity, AirlineNames),
GetWebSites(AirlineNames, WebSites)

The above mapping rule defines the relation **Airlines** through two virtual operations:
InquireAirlines returns the list of airlines operating between two cities, and **GetWebSites** returns the Web sites for a list of airlines.
The query infrastructure can obtain tuples of **Airlines** by invoking the operation **InquireAirlines** and then **GetWebSites**.

Users can directly use virtual operations to access Web services, but the use of relations has two benefits. For one thing, it lets users formulate and submit database-like queries in a natural way. For another, it provides a view tailored for a particular group of users interested in some specific part of the service space.

Virtual Operations Representation

Virtual operations represent functionalities a particular application domain typically offers. For any virtual operation appearing in a query, we need to locate the relevant Web service operations. This requires a semantic description as well as syntactic attributes. We assume that business partners would agree on a common ontology for describing concrete
Web services and virtual operations.

Each operation, whether virtual or concrete, is semantically described through its function and category.

Function contains two attributes:
• ***Functionality*** represents the business functionality provided by the operation (booking and listing, for example).
• ***Synonyms*** contains a list of alternative functionality names for the operation (for instance, reservation is a synonym of booking.).

Category also contains two attributes:

• ***Domain*** gives the area of interest of the operation (for example, hotel, accommodation, and tour).
• ***Synonyms*** resembles those defined for the operation's purpose.

To invoke any operation, we need to set values for its input variables. In formulating queries, users are free to specify any type of conditions on the different variables. This can lead to a scenario in which the system can't invoke operations because some input variables have missing values. If all virtual operation descriptions specified all their input variables' potential values, it would solve the problem (though this is not always possible for all input variables). The system could then expand an input variable in all its possible values that satisfy any condition appearing in the query. The following quintuple formally represents each virtual operation:

Vop = (In, Out, Domains, Category, Function)

where **In** is the set of input variables;
Out is the set of output variables;
Domains is a set of pair (**x,range**), where **x** is an enumerable variable appearing in **In** and **range** is the set of all possible values for **x**;
Category describes the domain of interest; and
Function describes the business function. The following is an example of a virtual operation with all of its attributes:

Fares = (In, Out, Domains, Category, Function)

where **In = (DepartureCity, ArrivalCity,DepartureDate)**,
Out = (DepartureDate, ArrivalDate, ArrivalTime, Price),

Domains= {(DepartureDate, [all dates between today's date and November 30, 2004])},
Category = {Flight,Trip}, and **Function = {Listing, Fares}**.

4.4 Multilevel Matching for Virtual Operations

As providers begin competing in their Web services offerings, differences will occur in the required input, returned output, QoWS, and so on. For example, a weather service's forecast details, freshness of information, and fees to access the service might vary depending on the provider. This means it will not always be possible to find an exact match for a given virtual operation. Instead of finding only concrete operations that match exactly the virtual operations appearing in a query, a more flexible matching scheme that allows virtual and concrete operations' attributes to differ might still satisfy users' needs.

We define a function **similar** to check whether two attributes appearing in two operations are the same: **similar**(*x*, *y*) is **True** if *x* and *y* correspond to the same concepts with respect to the common ontology defined in the application domain.

For any two operations *op*1 and *op*2, **In**(*op*1) = **In**(*op*2) if
In(*op*1) and **In**(*op*2) have the same number of variables and
Vx Ͼ **In**(*op1*) **(resp. In** (*op2*)**)**, ∃*y* Є
In(*op2*) **(resp. In**(*op1*)**)** | **similar**(*x, y*) **is True**

We define **Out**(*op*1) = **Out**(*op*2) similarly.
We also define **In**(*op*1) **Ç** **In**(*op*2) and **Out**(*op*1) C **Out**(*op*2) in a similar way, where the first set is a subset of the second one.

4.5 Matching the Operations

Let *Vop* and *Concop* be virtual and concrete operations, respectively. There are four different matching levels obtained by varying the way of attributes of virtual and concrete operations:
• exact match,
• overlapping match,
• partial match, and
• partial and overlapping match.

Exact match

In an *exact match*, the concrete operation matches the virtual operation in all attributes. Two operations match *exactly* if they have the same input and output variables, as well as the same **Category** and **Function**.

Overlapping match

An *overlapping match* relates to an operation offering *close* functionalities to that of the virtual operation. Two operations *overlap* if they have the same input and output variables, and their **Category** and **Function** overlap. Let us assume a virtual
operation
Fares = (In, Out, Domains, Category, Function)
where **In = (DepartureCity, ArrivalCity, DepartureDate)**,
Out = (DepartureDate, ArrivalDate, ArrivalTime, Price),
Domains = {(DepartureDate, [all dates between today's date and November 30, 2004])},
Category = {Flight,Trip}, and **Function = {Listing, Fares}**.

And a concrete operation
AirfareTickets = (In, Out, Category, Function)
where **In = (DepartureCity, ArrivalCity, DepartureDate)**,
Out = (DepartureDate, ArrivalDate, ArrivalTime, Price),
Category = {Flight, Trip, Tour, Tourism}, and
Function = {Listing, Fares, Quotes}
Provides an overlapping match.

Partial match

A *partial match* corresponds to the case where input and output attributes of the two operations do not coincide. Two operations *Vop* and *Concop* match partially if they have the same **Category** and **Function**, and **Out(Concop)** _ **Out(Vop)** or **In(Concop)** _ **In(Vop)**. An example of the first subset relationship is an operation that does not return all the output attributes the virtual operation expects. An example of the second subset relationship is two operations **fare1(<Travel Information>, IsStudent, Cost)** and **fare2(<Travel Information>, Cost)**. Both operations return fares for a trip specified by **<Travel Information>**, but the second one does not take into account whether the requester is a student, which might lower the price.

Combined partial and overlap match

A *partial and overlap match* combines the overlap and partial matches. Two operations *Vop* and *Concop* match partially and by overlap if **Out(Concop)** **_Out(Vop)** or **In(Concop)** _ **In(Vop)** and their **Category** and **Function** attributes overlap.

We assign a *matching degree* to each level that quantifies the precision of the matching. The matching degree directly affects the query results' quality. The above levels receive matching degrees of **1**, **3/4**, **1/2**, and **1/4**, respectively. The values are arbitrary, existing mainly to distinguish the different matching levels.

4.6 Query Processing

Users submit conjunctive queries (conjunctions of relations and conditions) over relations from the query level. Queries have the following general form:

$Q(X) :- \wedge_i R_i(X_i), \wedge_k C_k$

where *Ri* are relations from the query level.
X and *Xi* are tuples of variables; and
Ck represents conditions on variables appearing in the query.

The form of those conditions is *Ck = x op c*,
where *x* is an input or output variable appearing in any *Xi*,
c is a constant, and
op $\in \{=, \neq, <, >, \geq, \leq,\}$. Multiple occurrences of a variable express equality.

The query undergoes several transformations that result in a *service execution plan*, which basically defines a sequence of sets of operations. The query infrastructure can concurrently invoke operations within the same set. Optimization occurs at the Web service and data levels. The former concerns the choice of Web services and operations;
the latter relates to the ordering of operations invocations, data routing, and data manipulation operations for collecting results. Data-level optimization needs to ensure that the obtained ordering is feasible: whenever an operation appears in the ordering (that is, must be invoked), all its input variables must be bound (have a value).

The algorithm for this query infrastructure builds efficient service execution plans using a local selection approach for optimization. The algorithm selects the best concrete operation for each virtual operation appearing in a query (after the algorithm applies the mapping rules). To ensure the obtained plan's feasibility, the algorithm builds sequences of virtual-operation invocations iteratively on the basis of the availability of input variables for each operation. We assume that providers describe their Web services using WSDL augmented with the semantic attributes described in the "Virtual Operations Representation". We assume also that they publish them in UDDI service registries. Providers can advertise QoWS parameters using UDDI tModels. The main idea is that the algorithm builds sequences of virtual-operation invocations on the basis of input variables' availability. The algorithm then discovers concrete operations, matches them to virtual operations, and assesses them using the objective function while ensuring that the resulting sequence is still feasible.

The algorithm has three phases:

- initialization and query unfolding
- virtual operations ordering, and
- service discovery and operations matching.

Initialization and query unfolding

The *initialization and query unfolding* phase's main task is to collect all bound variables the algorithm obtains from the query either directly (equality conditions) or by using ranges (inequality conditions) defined in virtual operations. The algorithm also initializes the service execution plan to an empty set. It then unfolds the query to bear on virtual operations using the different mapping rules.

Virtual operations ordering

In the *virtual operations ordering* phase, the algorithm iteratively selects the virtual operations that it can invoke at a certain time in the sequence that represents the service execution plan. The algorithm makes its selection by determining, at each iteration, all virtual operations that have their input attributes bound — that is, whose variables the algorithm can replace with available constant values. Those selected virtual operations will eventually provide new bound variables through their output attributes. These would allow the algorithm to select other operations in the next iterations. If it finds an operation with no bound variables, the query is not answerable. At the end of this phase, the service execution plan bears on virtual operations. For example, assume that a query *Q* contains the relation **Airlines** defined in the "From Relations to Virtual Operations"

Q(HotelNames, WebSites) :- Hotel(City, bookingdate, hotelNames, WebSites), bookingdate = "10/02/06"

The algorithm replaces the relation **hotel** with the virtual operations **Inquirehotel** and **GetWebSites**. Based on bound variables **City** and **bookingdate**, the algorithm needs to select **Inquirehotel** first. The availability of the output attribute **hotelNames** would then allow the algorithm to select **GetWebSites** next in the sequence.

Service discovery and operations matching

The *service discovery and operations matching* phase's goal is to replace virtual operations with concrete operations, while making sure that the obtained sequence is still feasible based on available bindings at each position in the sequence. The algorithm traverses the sequence and initiates a lookup for each single virtual operation. The lookup should return a concrete operation with the highest value for the objective function, as defined previously. It starts by looking for relevant Web services through UDDI service registries, using the virtual operation's **Category** and **Function** attributes to build a UDDI inquiry.

The lookup searches the description of each returned Web service for operations that match the virtual operation using one of the four matching levels. For the previous query *Q*, several different providers could offer both virtual operations **InquireAirlines** and **GetWebSites** with different QoWS. The algorithm selects the concrete operation with the highest value for the objective function. Because a partial match can occur, the algorithm must check that there is no virtual operation in the sequence whose inputs depend on the missing outputs that a partial match allows. If the missing outputs are required, the algorithm executes the lookup again until it finds an appropriate concrete operation. The algorithm terminates when it has replaced all virtual operations in the sequence (service execution plan) with concrete operations, or when it determines that it can't match a virtual operation to any existing concrete operations.

CHAPTER V

IMPLEMENTATION

In order to test the proposed query infrastructure, it is implemented on top of Hotel reservation web service. The developed hotel reservation system has application such as listing of rooms available, reservation and cancellation of rooms, as Web services.

The Hotel reservation system is to be registered in UDDI registry. The application also have UDDI registry implementation prototype to check the query infrastructure. .Net and MS Access database is used as the UDDI registry.

The Hotel Application has three web services

1. **listhotel (city, fromdate, nodays, rate)**
2. **reservehotel (city, fromdate, nodays, rate)**
3. **cancelhotel (city, fromdate, nodays, rate)**

The Master tables are

1. Roomdetails
2. booking

Table : Roomdetails

Field name	**data type**
Roomnumber	number
Roomtype	text
City	text
Facility	text
Rate	currency

Table : Bookingdetails

Fieldname	**data type**
Bookid	number
Custid	number
Date	date/time
Nodays	number
Roomnumber	number

UDDI registry has a database for registering the service community. The registry has a table with following fields

Field name	**data type**
Registryno	number
Category(NAIS)	text
Service site	text
Synonyms	text
Technology	text
Reference	text
Owner/organisation	text
Address	text
Description	memo

Flow diagram

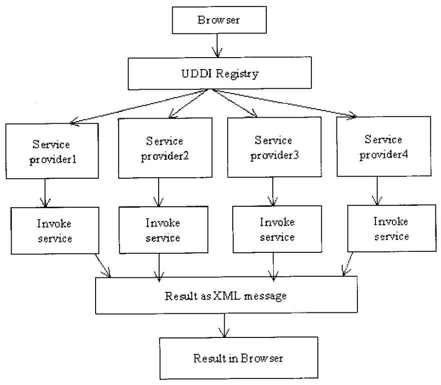
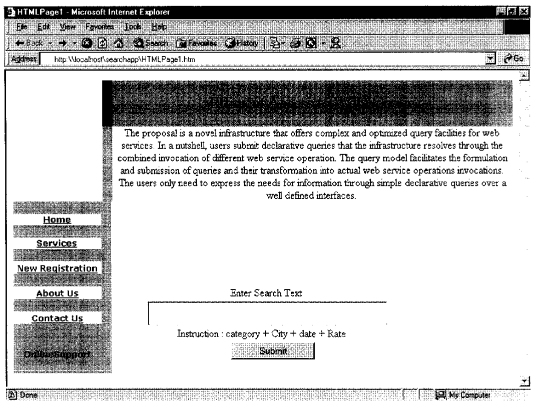


Figure 5.1 - Flow Diagram

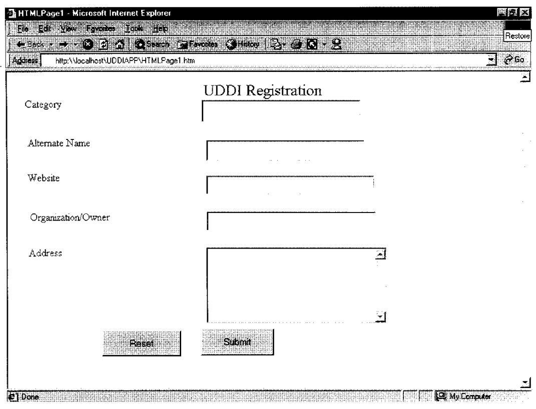
5.1 Screens


HTMLPage1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http:\\localhost\searchapp\HTMLPage1.htm
Go
Home
Services
New Registration
About Us
Contact Us
OnlineSupport
Enter Search Text
Submit
Done
My Computer

The proposal is a novel infrastructure that offers complex and optimized query facilities for web services. In a nutshell, users submit declarative queries that the infrastructure resolves through the combined invocation of different web service operation. The query model facilitates the formulation and submission of queries and their transformation into actual web service operations invocations. The users only need to express the needs for information through simple declarative queries over a well defined interfaces.

Instruction : category + City + date + Rate

Figure 5.1 Home Page for Search



HTMLPage1 - Microsoft Internet Explorer

Address: http:\\localhost\UDDIAPP\HTMLPage1.htm

UDDI Registration

Category	
Alternate Name	
Website	
Organization/Owner	
Address	

File Edit View Favorites Tools Help
Back Search Favorites History
Restore
Go
Reset
Submit
Done
My Computer

Figure 5.2 UDDI Registration Form

CHAPTER VI

CONCLUSION AND FUTURE OUTLOOK

Researchers in both academia and industry are studying Web services because of their role in deploying the Semantic Web. To the best the proposed query and optimization model is the first to provide complex query capabilities over Web services. It expresses queries over the query level and transforms them into a combination of virtual-operation invocations. Different matching modes match the virtual operations against concrete operations from actual Web services. The proposed model selects and combines appropriate operations based on matching degrees.

This query infrastructure can be extend several ways. Semantic-based optimization techniques for Web services would use intelligent agents to take advantage of the current context in order to enhance optimization.

K C T LIBRARY Coimbatore-641 006

APPENDIX

hotellisting web service

```
Imports System.Web.Services
Imports System.Data.OleDb
Imports System.Data

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
    Public Sub New()
        MyBase.New()
        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer
    'NOTE: The following procedure is required by the Web Services Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub
```

```
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the project.
    ' To test this web service, ensure that the .asmx file is the start page
    ' and press F5.
    '
    <WebMethod()> Public Function GetHotel(ByVal city As String, ByVal
fromdate As String, ByVal rate As Double) As DataSet
        Dim ds As New DataSet()
        Try
            Dim con As New OleDbConnection()
            Dim cmd1 As New OleDbCommand()
            con.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Password="""";User ID=Admin;Data
Source=C:\kumares" & _
            "h\sakthihotel.mdb;Mode=Share Deny None;Extended
Properties="""";Jet OLEDB:System d" & _
```

```
        "atabase="""";Jet OLEDB:Registry Path="""";Jet OLEDB:Database
Password="""";Jet OLEDB:E" & _
        "ngine Type=5;Jet OLEDB:Database Locking Mode=0;Jet
OLEDB:Global Partial Bulk Ops" & _
        "=2;Jet OLEDB:Global Bulk Transactions=1;Jet OLEDB:New Database
Password="""";Jet O" & _
        "LEDB:Create System Database=False;Jet OLEDB:Encrypt
Database=False;Jet OLEDB:Don" & _
        "'t Copy Locale on Compact=False;Jet OLEDB:Compact Without
Replica Repair=False;J" & _
        "et OLEDB:SFP=False"
        cmd1.CommandText = "select count(*) from hoteldet where city=? "
        cmd1.Parameters.Add("pcity", OleDbType.VarChar, 50).Value = city
        cmd1.Connection = con
        con.Open()
        Dim read1 As OleDbDataReader =
cmd1.ExecuteReader(CommandBehavior.SingleResult)
        If (read1.Read()) Then
            Dim c As Integer = read1.GetValue(0)
            If c <> 0 Then
                con.Close()
                Dim cmd2 As New OleDbCommand()
                cmd2.CommandText = "SELECT roomdet.*, hoteldet.* FROM
roomdet, hoteldet WHERE rno in ( SELECT roomdet.rno FROM roomdet
WHERE rno not in(SELECT roomdet.rno FROM booking INNER JOIN roomdet
ON booking.roomno = roomdet.rno WHERE (((booking.bookdate)=#" &
fromdate & "#))) and rate <" & rate & " ) "
                'cmd2.Parameters.Add("pdate", OleDbType.VarChar, 50).Value =
Format(fromdate, "MM/dd/yyyy")
                cmd2.Connection = con
                Dim adp As New OleDbDataAdapter()
```

```
                adp.SelectCommand = cmd2
                Dim c1 As New Data.Common.DataColumnMapping("rno", "rno")
                Dim c2 As New Data.Common.DataColumnMapping("roomtype",
"roomtype")
                Dim c3 As New Data.Common.DataColumnMapping("rate", "rate")
                Dim c4 As New Data.Common.DataColumnMapping("facility",
"facility")
                Dim c5 As New Data.Common.DataColumnMapping("hotelname",
"hotelname")
                Dim c6 As New Data.Common.DataColumnMapping("address",
"address")
                Dim c7 As New Data.Common.DataColumnMapping("city", "city")
                Dim cols() As Data.Common.DataColumnMapping = {c1, c2, c3,
c4, c5, c6, c7}
                Dim t1 As New Data.Common.DataTableMapping("Table",
"details")
                Dim ta() As Data.Common.DataTableMapping = {t1}
                adp.TableMappings.AddRange(ta)
                adp.Fill(ds)
                Return (ds)
            End If
        End If
    Catch ee As Exception

    End Try
    Return (Nothing)
End Function

End Class
```

REFERENCE

[1]. "Efficient Access to Web Services" – Mourad Ouzzani and Athman Bouguettaya, *IEEE Internet Computing, Vol 2, Apr 2004, Pg 34.*

[2]. "Service Discovery" – Steve Vinoski, *IEEE Internet Computing, Vol 1, Jan 2003, Pg 69.*

[3]. "Infrastructure for E-Government Web Services" – Brahim Medjahed, Rezgui, Athman Bouguettaya and Mourad Ouzzani, *IEEE Internet Comupting, Vol 7, no 1, Jan 2003, Pg58.*

[4]. "UDDI Technical White Paper", *Sep 2001, www.uddi.org*

[5]. "XML and DATABASE", *Tutorial, www.xml.org*

[6]. "Coordinating Business Transaction on Web", *IEEE Internet Computing, Vol 1, Jan 2003, Pg 30.*

[7]. "How Search Engine Works", Tutorial, www.searchenginewatch.com

[8]. www.oasis.org