1. INTRODUCTION

1.1 OVERVIEW

Web Service Standards, the SOAP and WSDL, have accomplished extraordinary interoperability across contrasting software systems. Therefore, broad range of new applications also tend to adopt Web Services because recent proposals for Web Service Standards have ensured extended features such as security, reliability, transactions, metadata management and orchestration.

A Web Service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another Web Service. Web Services are designed to be accessed by other applications and vary in complexity from simple operations, such as checking a banking account balance online, to complex processes running CRM (Customer Relationship Management) or ERP (Enterprise Resource Planning) systems (Erin Cavanaugh, 2006). Since they are based on open standards such as HTTP and XML-based protocols including SOAP and WSDL, Web Services are hardware, programming language, and operating system independent. This means that applications written in different programming languages and running on different platforms can seamlessly exchange data over intranets or the Internet using Web Services.

A Web Service can be identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web Service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols (Mohamed I. Mabrouk, 2008).
The basic three elements of Web Service based Systems are Service Provider or Service Repository, Service Requester and Service Registry (Chris Rayns et al, 2010).

- A Service Repository is responsible for building a useful Service, creating a Service description for it, publishing that Service description to one or more Service Registries and receiving Service invocation messages from one or more Service Requesters.
- A Service Consumer is responsible for finding Service description published to one or more Service Registries, and for using Service description to bind to or invoke Services hosted by Service Providers. Any Consumer of a Service can be considered a Service requester.
- The Service Registry is responsible for advertising Service descriptions publishes to it by Service Providers, and for allowing Service Requesters to search the collection of Service descriptions contained within the Service Registry. Once the Service Registry makes a match between the Service requester and Service Provider, the Service Registry is no longer needed for interaction (Steve Graham et al, 2002).

Universal Description, Discovery, and Integration (UDDI) is a specification designed to let businesses find each other and share business information. The UDDI Business Registry is a global, public, online directory that gives businesses a uniform way to describe their Services, discover other companies' Services, and understand the methods necessary to conduct e-business with a particular company (Abdaldhem et al, 2009).

The ability to form ad hoc federations of Services (peer services) or Directories promotes the formation of Service-Oriented syndicates. Service syndication seeks to promote in-demand Services by offering them throughout the federation, not just at one location (Darren Govoni, 2004). This also increases the reliability and availability of the Service, since it can be acquired and bound to from a variety of Providers. The incentive is such that the use of the Service benefits the syndicate members regardless of who provisions the Service, since that responsibility is now distributed.
There are certainly many issues that need to be resolved before peer technologies can exist peacefully within corporate boundaries and even more so for Web Services. Security is still the number one concern, as companies want to ensure that the forthcoming automation doesn't leave them vulnerable to malicious attack or unauthorized use (Darren Govoni, 2004).

This thesis advocates the use of a recently proposed distributed model for UDDI and proposes a methodology for Replica Management in the various nodes of the proposed Distributed UDDI.

Section 1.2 motivates the use of Web Services and Web Service based Systems for Business Applications. Section 1.3 elaborates the issues and challenges related to Web Service based Systems and Section 1.4 outlines the aims and objectives targeted in this work. Finally, Section 1.5 outlines the structure that this thesis follows to achieve these objectives.

1.2 WEB SERVICE SYSTEMS

This section introduces the Web Services and Web Service based Systems and its advantages. It elaborately discusses the various constituents of such systems, the Service Discovery, Service Provider and Service Consumer. This section also introduces the use of XML, HTTP, SOAP in Web Services. After establishing the introductory level discussions on Web Services and Web Service based Systems, this section discusses in detail the Web Service Standards and Service Oriented Architecture, which will give an insight of the way how the set of standards facilitate the Web Service based System developments.

1.2.1 Web Services

A Web Service (WS) is a (self-contained) software component that allows access to its functionality via a Web interface. Web Services communicate by
employing established protocols for message transport and encoding. Indeed, the W3C Web Services Architecture Working Group defines a Web Service as “A software application identified by an URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A Web Service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols” (Roland H. Schweitzer, 2003).

**Constitutes of Web Service based Systems:** A Web Service based System is composed of a Service Provider, Service Registry and a Service Requester (or) Service Consumer.

The services are created by the Service Provider and the created services are published. Services are published by posting the service information on the Service Directory. A Service Requester who is in need of a certain service, searches the Service Directory for one that meets the necessary criteria. If the Service Requester is able to find a service of his need on the service directory, the Service Provider can be requested for that service to fulfill the business need.

The Service Registry is for advertising service descriptions by Service Providers and for allowing Service Requesters to search the collection of service descriptions contained within the Service Registry. Once the Service Registry makes a match between the Service Requester and Service Provider, the Service Registry is no longer needed for interaction.

**1.2.1.1 Service Provider**

A Service Provider builds a useful service, creates a service description for it, publish that service description to one or more Service Registries and receiving service invocation messages from one or more Service Requesters. Service Provider is the service, the network-addressable entity, which can be a mainframe system, a component, or some other type of software system which can execute the requests
from Service Consumers. The Service Provider publishes its contract in the registry for access by Service Consumers.

In the SOA, a provider is the owner of a service. From a composite computing perspective, it is a software asset that others regard as a network-accessible service. In most cases, this software asset is exposed as a Web Service, which by definition has an XMLized description and a concrete implementation that encapsulates its behavior (David A. Chappell, 2002). Almost any piece of logic can be exposed as a service in an SOA, which may be a single component to a full-blown, mainframe-based business process, such as loan processing. Likewise, how the service is exposed is up to the provider. It can be accessed through SOAP over HTTP, through a JMS message queue, or via other technologies such as SMTP. The service may implement a request/response protocol, or it may just receive messages and deliver asynchronous replies.

As is often the case in modern software development, the term “Provider” contextually means either the organization which provides the service, the software itself, or the computer (or computers) on which the software runs. The Service Provider views the SOA as a framework for exposing its Web Services. These services are islands of code designed to solve one aspect of an overall business problem. A Service Provider ensures service availability, provides secure transaction environment and also provides quality of service as discussed further here.

- **Ensuring Availability:** A Web Service is not very useful if it isn't available. Making sure that a Web Service can accept service requests from a SOAP router is a paramount. The websites that host today's web applications have already figured out how to do this in a load-balanced, scalable way, and hence availability is ensured.

- **Providing a Secure Transaction Environment:** Most businesses already have security in place. However, the SOA presents interesting and nontrivial security problems. An SOA may encompass multiple sites, each with its
own way of implementing security. The challenge is to come up with a standards-based mechanism that allows each site in the SOA to propagate a security context, without necessarily having to use the same software. More likely than not, the two main security aspects, authentication and authorization will end up being Web Services themselves.

- **Quality of Service:** Web Services are an innovative and powerful new mechanism for heterogeneous distributed computing, but they still need to follow old "rules of conduct" to gain rapid, widespread acceptance. One of these rules guarantees a certain level of service.

  **Simple Object Access Protocol (SOAP):** Important component of Service Provider is Simple Object Access Protocol (SOAP). SOAP is a W3C recommendation. SOAP is an XML-based protocol for exchanging information between computers. SOAP is acronym for Simple Object Access Protocol. SOAP provides the data transport layer for Web Services which is designed for communication through internet and so, SOAP is a communication protocol. It is platform independent and can be used for broadcasting a message. SOAP is language independent too and it can exchange complete documents. SOAP enables client applications to easily connect to remote services and invoke remote methods. Other frameworks like CORBA, DCOM, and Java RMI, provide similar functionality but SOAP messages are written in XML.

  The use of SOAP can be explained as follows. To develop any application over internet, communication is necessary. Today's applications communicate using Remote Procedure Calls (RPC) but RPC brings in security problems. Firewalls and proxy servers do block this kind of traffic. Communication is established between objects, like DCOM and CORBA, but HTTP is not designed for such communications. HTTP communication is proved to be better between applications, because HTTP is supported by all Internet browsers and servers. So, SOAP provides facility to communicate between applications running on different operating systems, with different technologies and programming languages.
A SOAP message is a XML document containing an Envelope, a Header, Body and Fault as elements. Envelope defines the start and the end of the message. It helps to understand whether the message has been received or not and to indicate to be ready to process it. Header contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end point. can be used to specify a digital signature for password-protected services; likewise, it can be used to specify an account number for pay-per-use SOAP services. Header elements are optional part of SOAP messages and it can occur multiple times. These elements are intended to add new features and functionality and it is the first immediate child element of the SOAP envelope. If in a SOAP message more than one header is defined then all immediate child elements of the SOAP header are interpreted as SOAP header blocks. The Header element has two attributes which are the actor attribute and must understand attribute. Body element contains the XML data comprising the message being sent. It has the application-defined XML data being exchanged in the SOAP message. The body must be within the envelope and must follow any headers that might be defined for the message. The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema. Fault is an element that provides information about errors that occurred while processing the message. When an error occurs during processing this fault element is returned to the sender of the SOAP message. It is used as a mechanism to return specific information about error. SOAP message can carry only one fault block and that too is an optional part of SOAP message. Table 1.1 gives the sub-elements of the SOAP Fault element and the Table 1.2 presents the list of SOAP Fault Codes.

<table>
<thead>
<tr>
<th>Sub Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;faultCode&gt;</td>
<td>It is used to identify, based on the fault code, a class of errors.</td>
</tr>
<tr>
<td>&lt;faultString&gt;</td>
<td>It is used to explain the error type.</td>
</tr>
<tr>
<td>&lt;faultActor&gt;</td>
<td>It is used to indicate the cause of the error and this must be</td>
</tr>
</tbody>
</table>
communicated to the client. The Fault Actor element is included in all the nodes if the node is not the final destination node.

| <detail> | This is to carry the official application specific error messages. |

**Table 1.2 SOAP Fault Codes**

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VersionMismatch</td>
<td>Indicates if an invalid namespace is found for the SOAP Envelope element</td>
</tr>
<tr>
<td>MustUnderstand</td>
<td>Indicates that an immediate child element of the Header element, with the mustUnderstand attribute set to &quot;1&quot;, was not understood</td>
</tr>
<tr>
<td>Client</td>
<td>This is to indicate that the message was incorrectly formed or contain incorrect information</td>
</tr>
<tr>
<td>Server</td>
<td>Indicates that there was a problem with the server so the message could not proceed</td>
</tr>
</tbody>
</table>

**SOAP Encoding:** There available a set of rules for encoding data types using which the SOAP message can indicate a specific data type such as integers, floats, doubles, or arrays. SOAP data types are divided into two major categories as Scalar Types and Compound Types. Scalar types can contain exactly one value but Compound types can contain multiple values. Compound types are further subdivided into arrays and structs. SOAP arrays have a very specific set of rules which requires both the element type and the size of the array to be specified. It also supports multidimensional arrays but all SOAP implementations do not support multidimensional functionality. Type of Array and arrayType attribute (is required to specify the data type for the contained elements and the dimension of the array) are to be specified to create an array in SOAP. The following specifications are necessary to create an array. Also, SOAP Structs contain multiple values, but each element is specified with a unique accessor element.
**SOAP and HTTP:** SOAP is not tied to any one transport protocol. SOAP can be transported via SMTP, FTP or Microsoft Message Queuing (MSMQ). SOAP uses HTTP request to send SOAP request. SOAP responses are returned within the content of the HTTP response, which means SOAP specification includes its details on HTTP only. Both HTTP requests and HTTP responses are required to set their content type to text/xml. The SOAP specification mandates that the client must provide a SOAPAction header, but the actual value of the SOAPAction header is dependent on the SOAP server implementation.

1.2.1.2 Service Registry

A Service Registry is a network-based directory that contains available services. It is an entity that accepts and stores contracts from Service Providers and provides those contracts to interested Service Consumers. A Service Registry, which can be termed a Service Broker is the one who manages the Service Providers’ information such as the Services, its Description etc.,. This information includes business data such as name, description, and contact information, which are the white pages data. It also manages the green pages data like data describing policies, business processes, and software bindings which are needed to make use of the service. A Service Registry also offers intelligent search capabilities and business classification or taxonomy data, the yellow pages data. From a composite computing perspective, a broker represents a searchable registry of service descriptions, published by the Service Providers.

During the development cycle of a Web Service, a programmer uses the information in registries to create static bindings to services. At runtime, an application can tap into a registry (local or remote) to obtain service descriptions and create dynamic bindings to services.

Registries though often sound abstract, they solve a very concrete problem. They permit questions such as, "Who sells widgets?" If this question is answered,
more questions, such as, "How do I interact with their service to find prices, place orders, etc.?" can be answered. In short, a registry permits looking up a service and then to find its programmatic interface.

**Business Perspectives on the SOA:** A Service Registry is a business or software component whose main SOA-related activity involves maintaining service registries and their entries. Service Providers customarily pay registration fees to these brokers, who in turn advertise their service offerings. UDDI and ebXML Registries are the main "tools of the trade" for a service broker.

The kind of things a service broker looks into an SOA depends on the type of broker. The broker, who functions as a gateway is interested in finding other service registries. Gateways serve as a connection point to a network of external service registries. They differ from other service registries in that they are used primarily by the service registries themselves, as opposed to service requesters and providers. This being the case, gateways is primarily interested in finding other registries/brokers and expanding their reach. Other types of brokers concerned with locating and installing the documentation for Web Services. This activity is usually done on behalf of a Service Requester. It includes all the work required to obtain, install, version, and configure services before they are made available to clients. A brief discussion about UDDI registries follow and the detailed discussion is reserved to be discussed in Section 1.3.1.

UDDI (Universal Description, Discovery, and Integration) is a specification designed to let businesses find each other and share business information. The UDDI Business Registry is a global, public, online directory that gives businesses a uniform way to describe their services, discover other companies' services, and understand the methods necessary to conduct e-business with a particular company. Initial members include companies as diverse in their activities as Sun Microsystems, Ariba, Boeing, British Telecommunications, CBSI, Merrill Lynch, Descartes, Intel, IBM, Microsoft, and Fujitsu, to name only a very few (www.calsoftlabs.com/whitepaper).
UDDI Business Registry operators like IBM, Microsoft and Hewlett-Packard agreed to provide the means for companies to register their services, and for others to access them. The data is common to each Registry operator, to provide redundancy and to give worldwide access to everyone as quickly and easily as possible.

Published Web Services mean developers do not have to “re-invent the wheel” every time they want to do something that others have already done; for example getting a stock quote, tracking an en-route package, reading the news, or checking an airline reservation. Developers can discover and incorporate prewritten Web Services within their own applications quickly and easily using the UDDI registries through the internet.

Anyone can search the registry, but a registration is required to publish information about a company and its services. The companies can provide links to web sites, ftp areas, email and postal addresses and more importantly describe the services provided by the company and also the methods by which those services can be accessed. WSDL, Web Services Description Language is a major component of a service repository, which is discussed in the section to follow.

**WSDL (Web Services Description Language)** defines the XML grammar for describing services as collections of communication endpoints capable of exchanging messages. Companies can publish WSDLs for services they provide and others can access those services using the information in the WSDL. Links to WSDLs are usually offered in a company’s profile in the UDDI registry.

WSDL defines some XML grammar for describing communications regarding Web Services in a structured and standardized way. It is also extensible and allows using other type definition languages. A WSDL document defines services as collections of network endpoints, or ports, no matter what message formats or network
protocols are used to communicate. Ports are capable of exchanging messages, which are defined as abstract descriptions of the data being exchanged.

A WSDL message uses number of elements to define Web Services (www.w3.org) which are Types, Message, Operation, Port Type, Binding, Port and Service. Types are the containers for data type definitions using some type system. Message is an abstract, typed definition of the data being communicated. Operation is an abstract description of an action supported by the service. Port Type is an abstract set of operations supported by one or more endpoints. Binding is a concrete protocol and data format specification for a particular port type. Port is a single endpoint defined as a combination of a binding and a network address and Service is a collection of related endpoints.

1.2.1.3 Service Consumer/Requester

A Service Consumer who is in need of a service finds the service description published to one or more Service Registries, and using the service description to bind to or invoke services hosted by Service Providers. Any consumer of a service can be considered a Service Requester. The Service Consumer is an application, service, or some other type of software module that initiates the locating of the service in the registry, binding to the service over a transport, and executing the service function. The service consumer executes the service by sending it a request formatted according to the contract.

In Web Service based Systems, a requester is a business that discovers and invokes software assets provided by one or more providers. From a composite computing perspective, a requester is an application that looks for and initiates an interaction with a provider. This role could be played by a person using a web browser or by computational entities without a user interface, such as another Web Service. Depending on the context, a requester can be a person, an organization, or a piece of software.
A Service Requester typically views an SOA as something it uses to access the Web Services that provide it with the data it gives to its customers. Ideally, these Web Services allow the business to receive this data in an exact format or structure, thereby eliminating the need for elaborate data integration or mapping.

Major components of Service Consumer are Extensible Markup Language and Hypertext Transport Protocol.

**Extensible Markup Language (XML):** Extensible Markup Language (XML), which has become the common language of the computing business, is a tag-oriented language that looks superficially like HTML, but its purpose is different. HTML describes the way how a document should look like when displayed in a browser. Whereas, XML can describe the meaning of the given data, independent of the way that it is displayed. Briefly, XML is to remove the ambiguity from data.

Adding these XML style tags to the data makes it easy to parse and use. An additional feature of XML is the existence of a description of what the tag structure must look like in a document for it to be valid. The original description was called the Document Type Description (DTD), but it is fast being made obsolete by the XML schema. They both serve the same purpose, but the XML schema is more powerful and allows for more precise descriptions.

**Hypertext Transport Protocol:** Hypertext Transport Protocol (HTTP) is the workhorse of the Web. It is managed by the W3C as well. Its current version is 1.1, and all activity has ceased on it because it is considered complete and stable. The purpose of HTTP is to provide a protocol to move requests and responses between clients and servers. HTTP carries any information that is placed in it from point A to point B without regard for its data type. As a result, it is a popular way to transport SOAP messages between clients and Web Services.
1.2.2 Web Service Standards

Web Service technology has introduced a new abstraction layer and a radically new architecture for software. By employing a set of XML standards to define and describe Web Service functionalities, several tasks such as discovery and composition of these services are facilitated (or even automated) to some extent. Web Service technology also aims to facilitate the interaction between different Web Services (i.e., software programs) by enforcing the use of XML standards for data exchange. Note, that any kind of data can be exchanged between Web Services (e.g., semi-structured, textual, structured) as long as it is embedded in an XML based messaging protocol. Figure 1.1 (de Aalst, 2003) shows the main Web Service technology standards, all based on XML. A Web Service interface is described using the Web Service Description Language (WSDL).

![Figure 1.1 Web Service Standards](image)

Web Services exchange messages encoded in the SOAP (Simple Object Access Protocol) messaging framework and transported over HTTP or other Internet protocols. Several tasks can be performed with Web Services. A typical Web Service life-cycle envisions the following scenario. A service provider publishes the WSDL description of his service in UDDI, a registry that permits Universal Description Discovery and Integration of Web Services. Subsequently, Service Requesters can inspect UDDI and locate/discover Web Services that are of interest. Using the
information provided by the WSDL description they can directly invoke the corresponding Web Service. Further, several Web Services can be composed to achieve a more complex functionality. Such compositions of services can be specified using Business Process Execution Language for Web Services (BPEL WS). By relying on these standards, Web Services hide any implementation details therefore increasing cross-language and cross-platform interoperability.

**Example Scenario:** Many Web Services allow access to large databases permitting controlled access to information that might not be explicitly stated on Web pages. For example, the pharmacy finding process could be repeated for any zip code (or for any health care provider) relying on the output of one or more Web Services and not on data provided by static Web pages. A good Web Service to generalize this task is the Medicare Supplier Web Service which can retrieve details of Medicare suppliers given a zip code, a city name or the types of supplies provided.

Figure 1.2 shows a schematic representation of the WSDL file associated to the MedicareSupplier service. WSDL has considerable support from industry and increasing tool-support (WSDL generators, editors). As an XML-based language, it is machine processable, and a structured and standardized way to describe web-interfaces of services. In WSDL a service is seen as a collection of network endpoints which operate on messages. The example service provides one port: MedicareSupplierSoap. This port groups together three operations that return lists of Medicare suppliers and their details given a zip code (for GetSupplierByZipCode), a city name (for GetSupplierByCity) or the description of the supplied material (for GetSupplierBySupplyType). Each operation has an input (IMsg) and an output (OMsg) message. A message has a name and a set of parts of certain type. Parts represent input/output parameters depending if they are declared in the input or the output message.
Service(
    PortType: MedicareSupplierSoap ( 
    op:GetSupplierByZipCode(
        IMsg(zip), OMsg(SupplierDataLists))
    op:GetSupplierByCity(
        IMsg(City), OMsg(SupplierDataLists))
    op:GetSupplierBySupplyType(
        IMsg(description), OMsg(SupplierDataLists))
) )

Figure 1.2 Sample Schematic Representation of a WSDL File

The example above does not state the name of the message but gives only the name of its parts. The type of the parts can be any XML Schema data type or a previously defined complex type. A WSDL document has two major parts. First, the abstract interface of the service specifies the data types, messages and port Types with the corresponding operations (which refer to previously defined messages). Second, an implementation part binds the abstract interface to concrete network protocols and message formats (SOAP, HTTP).

1.2.3 Service Oriented Architecture

It has become startlingly clear that the viability of a business requires adaptability, accountability and credibility. All Businesses, now a days, rely on technology to become more flexible and they need to anticipate and plan for the future for which right decisions at right time is the need of the day. Today, the very survival of a business pivots on its ability to adapt its IT to meet every-changing business challenges. Service Oriented Architectures (SOA) help business and IT to unify goals and bridge the gaps between their very separate worlds by establishing a common language and creating a more flexible infrastructure to support change.
SOA is a business approach to building IT systems which allows businesses to Leverage existing assets, create new ones, easily enable the inevitable changes required to support the business. Building and deploying IT systems is made easier through SOA that directly serve the goals of a business. Change of business is made easy through SOA because it integrates business requirements with an IT framework that simultaneously makes it possible to leverage existing systems and enables business change. SOA enables the business to keep its focus on business and allows IT to evolve and keep pace in a dynamically changing world.

SOA helps building business applications as a set of loosely coupled black box components. These components are further orchestrated in such a way to deliver well-defined level of service by linking together business processes. SOA is meant for building business applications. SOA hides the complexity behind the development of individual loosely components but only permits the use of those components to build the required business applications. Every component is autonomous here and each component just passes data to another component and interact. Each component offers a small range of simple services to other components. Complex business applications are arrived by way of orchestration of different components.

1.3 Limitations of Web Service Systems

Based on the discussions through the sections above, it is clearly understood that the Web Services and Web Service based Systems have got lot of advantages like simplicity, modularity, scalability, easy to build etc.,. Though the Web Services have remarkable benefits, it has its own pitfalls which are listed below.

- The simplicity of Web Services which was considered to be an advantage can also be a hindrance. Web Service requests are larger than requests encoded with a binary protocol which becomes an issue under lower bandwidth connectivity or during the time when the network is busy.
The core Web protocols, HTTP and HTTPS are simple but not meant for long term sessions. In a CORBA or RMI type of environment, a client stays connected for an extended period of time and exchanges data with the server. Such interactions are difficult with Web Services.

The problem with HTTP and HTTPS when it comes to Web Services is that these protocols are "stateless"—the interaction between the server and client is typically brief and when there is no data being exchanged, the server and client have no knowledge of each other. More specifically, if a client makes a request to the server, receives some information, and then immediately crashes due to a power outage, the server never knows that the client is no longer active. The server needs a way to keep track of what a client is doing and also to determine when a client is no longer active.

Typically, a server sends some kind of session identification to the client when the client first accesses the server. The client then uses this identification when it makes further requests to the server. This enables the server to recall any information it has about the client. A server must usually rely on a timeout mechanism to determine that a client is no longer active. If a server doesn't receive a request from a client after a predetermined amount of time, it assumes that the client is inactive and removes any client information it was keeping. This extra overhead means more work for Web Service developers.

In addition to the above, for the Web Services to succeed there are many other technical challenges that have to be met. Some of the issues are Reliability, Security, Transactions, Scalability, Manageability, Accountability and Testing. Above all, the issue which is very relevant to this thesis is related to Web Services Discovery. Which is about how a Web Service advertise itself for discovery by other services. After the services advertisement, what happens if the services changes or moves is a question to be resolved, though some solutions are provided by researchers as found in Chapter 2.
1.3.1 Universal Description, Discovery and Integration (UDDI)

UDDI (Universal Description, Discovery, and Integration) is a specification designed to let businesses find each other and share business information. The UDDI Business Registry is a global, public, online directory that gives businesses a uniform way to describe their services, discover other companies' services, and understand the methods necessary to conduct e-business with a particular company. Initial members include companies as diverse in their activities as Sun Microsystems, Ariba, Boeing, British Telecommunications, CBSI, Merrill Lynch, Descartes, Intel, IBM, Microsoft, and Fujitsu, to name only a very few.

UDDI registries by IBM, Microsoft and Hewlett-Packard are so popular. These companies simply agree to provide the means for companies to register their services, and for others to access them. The data is common to each Registry operator, to provide redundancy and to give worldwide access to everyone as quickly and easily as possible. Published Web Services mean developers do not have to “re-invent the wheel” every time they want to do something that others have already done; for example getting a stock quote, tracking an en-route package, reading the news, or checking an airline reservation. Using the Internet, developers can discover and incorporate prewritten Web Services within their own applications quickly and easily. Anyone can search the registry, but to publish information about a company and services, one must register. Information to describe the services provided and methods by which they can be accessed can be entered (www.enterprisecomponent.com).

**UDDI Elements:** A business or company can register three types of information into a UDDI registry. This information is contained into three elements of UDDI. These three elements are White Pages, Yellow Pages and Green Pages. White Pages contains basic information about the company and its business, basic contact information including business name, address, contact phone number etc.,. Also it carries an unique identifiers for the company tax IDs which allows others to discover the Web Service based upon the business identification. Yellow Pages contain more
details about the company, and include descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it. It uses commonly accepted industrial categorization schemes, industry codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want. Green Pages contains technical information about a Web Service. This is what allows someone to bind to a Web Service after it's been found. This includes the various interfaces, the URL locations, the discovery information and similar data required to find and run the Web Service.

**UDDI Data Model:** UDDI includes an XML Schema that describes five data structures which are `businessEntity`, `businessService`, `bindingTemplate`, `tModel`, `publisherAssertion`. The `businessEntity` structure represents the provider of Web Services. Within the UDDI registry, this structure contains information about the company itself, including contact information, industry categories, business identifiers, and a list of services provided. The `businessService` structure represents an individual Web Service provided by the business entity. Its description includes information on how to bind to the Web Service, what type of Web Service it is, and what taxonomical categories it belongs to. `bindingTemplates` are the technical descriptions of the Web Services represented by the business service structure. A single business service may have multiple binding templates. The binding template represents the actual implementation of the Web Service. `tModel` stands for technical model, which is a way of describing the various business, service, and template structures stored within the UDDI registry (www.courses.csusm.edu).

Any abstract concept can be registered within UDDI as a `tModel`. For instance, if a new WSDL port type is defined, a `tModel` can be defined which represents the port type within UDDI. Then, a given business service implements that port type by associating the `tModel` with one of that business service's binding templates. `publisherAssertion` is a relationship structure putting into association two or more `businessEntity` structures according to a specific type of relationship, such as subsidiary or department. This structure consists of the three elements from `Key` (the
first businessKey), toKey (the second businessKey) and keyedReference. The keyedReference designates the asserted relationship type in terms of a keyName keyValue pair within a tModel, uniquely referenced by a tModelKey.

**UDDI API Specification:** This is a Specification of API for searching and publishing UDDI data. The specifications include:

- **UDDI Replication:** It describes the data replication processes and also the interfaces. These interfaces are registry operator to conform achieving data replication among different sites.
- **UDDI Operators:** This document outlines the behavior and operational parameters.
- **UDDI Programmer's API:** This specification inquiring about services hosted in a registry.
- **UDDI Data Structures:** Specification of XML structures contained within the SOAP messages.

**UDDI Interfaces:** A registry without a way of accessing it is of no use. The UDDI standard version 2.0 specifies two interfaces for Service Consumers and Service Providers to interact with the registry. Service Consumers use Inquiry Interface to find a service, and Service Providers use Publisher Interface to advertise a service. The core of the UDDI interfaces is the UDDI XML Schema definitions. These define the fundamental UDDI data types through which all the information flows.

**The Publisher Interface:** The Publisher interface defines sixteen operations for a service provider to manage the entries in the UDDI registry, which are discussed as given below.

- **get_authToken:** Retrieves an authorization token. All of the Publisher interface operations require that a valid authorization token be submitted with the request.
- **discard_authToken**: Tells the UDDI registry to no longer accept a given authorization token. This step is equivalent to logging out of the system.

- **save_business**: Creates or updates a business entity's information contained in the UDDI registry.

- **save_service**: Creates or updates information about the Web Services that a business entity provides.

- **save_binding**: Creates or updates the technical information about a Web Service's implementation.

- **save_tModel**: Creates or updates the registration of abstract concepts managed by the UDDI registry.

- **delete_business**: Removes the given business entities from the UDDI registry completely.

- **delete_service**: Removes the given Web Services from the UDDI registry completely.

- **delete_binding**: Removes the given Web Service technical details from the UDDI registry.

- **delete_tModel**: Removes the specified tModels from the UDDI registry.

- **get_registeredInfo**: Returns a summary of everything the UDDI registry is currently keeping track of for the user, including all businesses, all services, and all tModels.

- **set_publisherAssertions**: Manages all of the tracked relationship assertions associated with an individual publisher account.

- **add_publisherAssertions**: Causes one or more publisherAssertions to be added to an individual publisher's assertion collection.

- **delete_publisherAssertions**: Causes one or more publisherAssertion elements to be removed from a publisher's assertion collection.

- **get_assertionStatusReport**: Provides administrative support for determining the status of current and outstanding publisher assertions that involve any of the business registrations managed by the individual publisher account.

- **get_publisherAssertions**: Obtains the full set of publisher assertions that is associated with an individual publisher account.
The Inquiry Interface: The inquiry interface defines ten operations for searching the UDDI registry and retrieving details about specific registrations as given below.

- **find_binding**: Returns a list of Web Services that match a particular set of criteria based on the technical binding information.
- **find_business**: Returns a list of business entities that match a particular set of criteria.
- **find_itservice**: Returns a list of Web Services that match a particular set of criteria.
- **find_tModel**: Returns a list of tModels that match a particular set of criteria.
- **get_bindingDetail**: Returns the complete registration information for a particular Web Service binding template.
- **get_businessDetail**: Returns the registration information for a business entity, including all services that entity provides.
- **get_businessDetailExt**: Returns the complete registration information for a business entity.
- **get_serviceDetail**: Returns the complete registration information for a Web Service.
- **get_tModelDetail**: Returns the complete registration information for a tModel.
- **find_relatedBusinesses**: Discovers businesses that have been related via the uddi-org:relationships model.

1.3.2 Challenges of UDDI

UDDI enables businesses to publish their Web Services and search them based on consumer preferences. The present centralized UDDI structure is less robust and it is difficult to support a large number of Web Services. Centralized UDDI registries collect the Web Services information in a passive manner. The real-time validity of the
services information cannot be guaranteed in a Centralized UDDI because it waits for service publication, update or discovery request passively (R. Rajmohan et al, 2011).

UDDI specific systems have to guarantee real time responses to service enquiries with the services’ data that reflects the most updated information of services. But they do not have an automatic mechanism for updating the registry as and when there are changes in the services’ information (Qianhui Liang et al, 2009).

The current UDDI attempts to reduce the disadvantages of the centralized approach by replicating the entire information on different registries. It temporarily improves the performance of UDDI but it involves a high cost in deployment and maintenance. Several decentralized approaches have been proposed moving from a central design to a distributed structure by connecting registries with distributed technologies (R. Rajmohan et al, 2011).

Also, centralized UDDI induces performance bottleneck, restricts the scalability of the environment, and also would become a cause for single point of failure. This section considers only the problems related to the Centralized approach of UDDI. Other than the problems discussed here with respect to a Centralized UDDI, UDDI also suffers from the problems of service discovery only based on functional requirements, failure during keyword search to recognize the similarities and differences between the capabilities provided by Web Services, incapable of providing Quality of Service (QoS) measurements for the registered Web Services, lack of guarantees to the validity and quality of information it contains etc., Following section discusses the UDDI in a distributed approach (Khaled Ragab, 2008) (Eyhab Al Masri et al, 2008) (Qusay H. Mahmoud et al, 2007)(www2008.org).

1.3.3 Distributed UDDI (DUDDI)

Centralized UDDIs are suffering from the problems of single point of failure, performance bottleneck and scalability issues other than being passive in collecting the information related to the change of state of published Web Services. Researchers
proposed a distributed architecture for UDDI to reduce the said problems due to centralized UDDI approach. Following features are expected to hold good in a Distributed UDDI environment (Reaz Ahmed, 2010).

**Salient Features of DUDDI:** A list of some of the essential features of a Distributed UDDI is given as follows.

- **Efficiency:** A Distributed UDDI supports publish and retrieval of Web Services’ information without consuming significant resources.
- **Scalability:** Searching efficiency in DUDDI does not degrade with increase in network size or the volume of Web Services registered because of the number of available registry can be increased.
- **Search Completeness:** DUDDI offers maximum discovery of requested objects with high precision and recall.
- **Fault Resilience:** DUDDI ensures services availability and reduces the overhead of node failure.
- **Load Balancing:** Distribute routing, storage and processing loads according to the capabilities of the participating nodes is possible in a DUDDI.

Eyhab Al-Masri et al (2007) suggest multiple business registries for the UDDI for the following reasons: The UDDI does not provide any guarantees to the validity and quality of information it contains, 2. The disconnection between UDDI and the current Web increases the chances of getting inappropriate results for a Web Service Discovery queries and 3. The UDDI is incapable of providing Quality of Service (QoS) measurements for registered Web Services, which would offer, if available, helpful information to clients when choosing appropriate Web Services.

Ni Yulin, Si Huayou et al (2010), who proposed PDUS, P2P based DUDDI Service Discovery Approach, writes that in a traditional Web Service discovery method, Service Providers and Requesters publish or query Web Services through a centralized registry, UDDI, which suffers from single point of failure and many performance bottlenecks such as too many data in single centralized UDDI which is
difficult to maintain and too frequent registrations and lookup on a single registry introduces lot of challenges in servicing the requests from the clients.

1.4 MOTIVATIONS

This section discusses the need for a research contribution in the domain of Distributed UDDI for whatever the issues and challenges of Centralized UDDI, discussed in the earlier sections. For the said problems of Centralized UDDI, a distributed approach was presented for Web Services discovery process but while the Distributed Approach solves the major issues of Centralized UDDI, DUDDI itself introduces research challenges. This section introduces such challenges and that is the motivation for taking up this research.

1.4.1 Issues and Challenges of Distributed UDDI

Eyhab Al et al (2007) recommends multiple business registries but says that Web Services proliferate and the searching for specific business applications or Web Services become highly challenging and time consuming particularly as the number of UDDI Business Registries (UBRs) begins to multiply. The authors further add that decentralizing UBRs add another level of complexity of how to effectively identify that registry which keeps the updated and current version of Web Services from these distributed multiple registries, which would have been easier if the UDDI was centralized otherwise.

Distributed UDDI registries are deployed across various regions. Exchange of information from one UDDI registry to another is to be managed. Each UDDI registries may belong to different businesses and thus would introduce different requirements on acceptable announcements and retrieval requests of Web Services. Also the UDDI registries are not interlinked with a predefined communication channels. In the P2P Distributed UDDI setup, there is no predefined centralized component, which manages the UDDI registries. Each UDDI registry is independent
in defining the announcements of providers that it accepts and the retrieval requests of users that it satisfies. The definition is based on UDDI registry-defined policies. In addition each provider is independent in selecting the UDDI registries to which it would like submitting its announcements of Web Services (W. Mansoor, 2004).

Kunal verma et al (2005) says that dealing with multiple registries, hundreds in number, is a challenging task during service publication, updation and discovery. He notes that locating an appropriate registry for the correct version of the required Web Service is a critical task to be resolved. If the registries are categorized according to the domain of business, identification of proper registry would be easier, author adds in his paper.

In the context of Web Service Publish in a Distributed UDDI, the published Web Services information are propagated to a cluster of registries to facilitate faster service recovery by the clients which reside in a highly dispersed locations. As and when the service provider updates the published Web Services information it needs to be communicated to all the registries wherever the corresponding information is maintained. There are chances that the information of a same Web Service would be available in different versions in different registries. That leads to Web Service information inconsistencies. It becomes a challenging task for the client to identify the right registry to acquire the current version of a particular Web Services which he intends to retrieve.

Also, replication of Web Services information in different registries requires an efficient and error free methodology so as to ensure the availability of updated and current version of Web Services definition in the DUDDI registries. It is a challenging task to identify the number registries which are to be updated on a particular Web Service’s information.
1.4.2 Need of Replication Management Methodologies in DUDDI

This section presents the motivations and the need for Replica Management Methodologies in Distributed Universal Description, Discovery and Integration (DUDDI). In particular, the discussion is focused on the problems currently experienced in DUDDI and highlights how the problem of managing the replicas in various nodes at which the Web Services definitions are being maintained.

Many of the efforts that attempted to improve the discovery of Web Services have failed to address the issue across multiple UBRs. Due to the fact that UBRs are hosted on Web servers, they are dependent on network traffic and performance, and therefore, clients that are looking for appropriate Web Services are susceptible to performance issues when carrying out multiple UBR search requests.

Many researchers have focused on discovering Web Services through a centralized UDDI registry (Qusay H. Mahmoud, 2007). But research to identify methodologies for Web Services Information replication across different Distributed UDDI registries is highly felt because the centralized UDDI registry suffers from the problems discussed in the earlier sections and brought in the necessity of moving forward with Distributed UDDI registries. When it is highly agreed that the UDDI registry need to be distributed, Services Information replication across different registry nodes becomes extremely vital and necessary.

Web Services information replication across different DUDDI registry is not only necessary but also immensely benefit the clients when they approach the DUDDI registries for Service recovery. In a Centralized UDDI structure, clients need to recover the Services from the Central registry which, in normal case, would incur a huge cost in terms of data transportation from the remote registry to the client and also the registry might be busy because the UDDI registry, being centralized, would be accessed by many client simultaneously. So, replication of Web Service information to a cluster of DUDDI registries would ensure that the clients would get the required
service from a registry which might be in a very close proximity to the requested client (Lukasz Juszczyk et al, 2006). Also, because the Services information is replicated in many nodes, all requesters need not throng in a single registry which will avoid traffic routed to single node.

Since replication methodology in a Distributed UDDI proves to be a highly necessary and beneficial research issue, it becomes the central point of motivation for initiation of a research in this thesis. The rest of the thesis will discuss issues and proposals related to Web Services information replication in a Distributed UDDI environment.

1.5 CONTRIBUTION TO KNOWLEDGE

Based on the discussions, the Distributed UDDI system needs methods and means for the replica management and other challenges identified. This research has its objectives to offer an enhanced model for DUDDI registry with improved performance in terms of availability of the service information associated with DUDDI, overall consistency of entries of the registries and service search time in the distributed environments. This research aims to contribute its outcome in the domain of Distributed UDDI registry

1. By providing a robust Replica Management technique to ensure the Maximum Availability of the service registries’ information in the DUDDI environments.
2. By improving the consistency of the replicas of the service registries’ information in the DUDDI environments.
3. By minimizing the Service Search Response Time in the DUDDI environments.

For achieving an effective model for a Distributed UDDI system and its replica management methodology, the proposed research has the following contributions as listed below.
• A Systematic Model for Distributed UDDI Environment is being proposed and realized with essential characteristics of the service computing era.
• A Testbed for the DUDDI Environment is being designed and to be established to exhibit the Real World scenario.
• A Methodology for Service Replica Management in the DUDDI Environment is being designed and to be implemented.
• A set of appropriate experiments have been designed and to be performed to prove the claims made in the proposed research.
• The outcomes of the proposed research are to be assessed and validated using the identified performance evaluation criteria.

1.6 ORGANIZATION OF THE THESIS

Chapter 1 provides the introduction of the thesis. Chapter 2 discusses the related work done so far in the field of Web Service based Systems and more particularly in the identified area of UDDI. It also figures out the issues and status of DUDDI paradigm in Service Oriented Architecture. Chapter 3 describes the goals of the research and the Experimentation Methodology applied in this research. Chapter 4 focuses on Modeling and Simulation of a Distributed UDDI environment. Chapter 4 also elaborates the proposed methodology for Web Services Replica Management methodologies in the DUDDI registries. Chapter 5 deals about the methodology proposed for the Service Replica Management, which is about the methodologies for Service/Replica Placement and Service/Replica Retrieval in DUDDI registries. Chapter 6 analyses the performance of the proposed DUDDI Model through the analysis of the experimental results achieved for Replica Management based on the methodology defined in Chapter 5. Chapter 7 provides conclusive statements and future enhancements on this research.