Chapter 1

Introduction

1.1 Service Oriented Architecture

Service-oriented architecture is about the evolution of business processes, applications and services from today's legacy-ridden and smooth integration of disparate applications to a world of connected businesses, accommodating rapid response to change and utilizing vast degrees of business automation. It is a set of general design principles that enables organizations to change business processes on the wing and respond to the shifting demands of the business in a manner that would be impractical or cost-prohibitive using conventional application development and resources allocation [29]. SOA can be viewed as a computing methodology or approach to building IT systems in which business services i.e. services provided by an organization to clients are the key organizing principles used to align IT systems with the needs of the business. Earlier approaches used in building IT systems focused on direct use of specific implementation environments such as object orientation or procedure orientation to solve business problems. These approaches resulted in systems that are often tied to the features and functions of a particular execution environment technology. From the above description of service-oriented architecture, it shows clearly that service is a key component. A service can be considered as a means by which the needs of a consumer are brought together with the capabilities of the service provider [68]. Services within an organizational context can either be driven by the needs of the consumer, user/business requirements and drilling down to the system level (top down), or taking into account the system capabilities of the service provider and building up services that can be exposed to the higher layers in
the architecture (bottom up). But as it is today, services are more built from the engineers or suppliers view than that of the users.

The interest in SOA as a guiding principle was as a result of the IT community shifting away from large scale development of applications and towards the creation of services that more accurately reflect underlying business processes [68]. The business and IT sector now complement and needs each other than ever before. But over the years, the successful integration of these two sectors has been a nightmare even with the emergence of different technologies. While previous technologies have not efficiently enabled the IT/business-unit relationship to improve, it is the belief of researchers and IT professionals that the nature of services as a consumable product represents what could be the much needed shift. The major difference between service-oriented development and previous approaches is that service orientation focuses on the description of the business problem, while previous approaches focus more on the use of a specific execution environment technology. The technique with which services are developed enhanced their alignment to solving business problems than was the case with previous generations of technology.

![Service Oriented Architecture](image)

*Figure 1.1 Service Oriented Architecture*
A Service Oriented Architecture contains three basic components as depicted in Figure 1.1. A service provider: The service provider creates a Web service and possibly publishes to the service broker the information necessary to access and interface with the Web service. A service broker: The service broker (also known as a service registry) makes the Web service access and interface information available to any potential service requestor. A service requestor: The service requester binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

Each component can also act as one of the two other components. For example, if a service provider requires information that it can only acquire from some other service, it acts as a service requester while still serving the original request [89].

The service provider creates a Web service and possibly publishes its interface and access information to the service broker. The service broker (also known as service registry) is responsible for making the Web service interface and implementation access information available to any potential service requestor.

The service requestor binds to the service provider in order to invoke one of its Web services, having optionally placed entries in the broker registry using various find operations.

1.1.1 Benefits of SOA

The major reason for the emergence of SOA is for the relationship between IT and the business units to improve. Business organizations are dealing with two fundamental concerns; the ability to quickly change to meet today’s urgent demand
for new level of agility, and the need to reduce cost [68]. To remain competitive, businesses must adapt quickly to internal factors such as acquisition and restructuring, or external factors like competitive forces, customer requirements or government regulations. Cost-effective, flexible IT infrastructure is highly needed to support the business. The concept of service oriented architecture can help organizations succeed in the dynamic business landscape of today. This can be achieved through the primary characteristic of SOA which encourages the reuse of business logic. SOA, when properly implemented, makes reusability extremely cost-effective. The motivations for different service oriented architecture initiatives include a range of technical and business reasons. The most common motivations are agility, flexibility, reusability, data rationalization, integration, and reduced costs.

The existing mainframe systems also called legacy systems are not capable of some of the mentioned benefits. For example, the cost of maintenance a legacy system is very expensive and does not support critical business application because business logic and data are not separated.

1.1.2 Challenges of SOA

The research work like to stress this point that an SOA is not a silver-bullet solution for all problems when it comes to IT and business integration. Implementing an SOA is good but should not be expected to suit every IT and business domains, sometimes a different kind of architecture is more suited to solve some problems. For example if you have hard real-time or near-real-time requirements, an SOA approach always introduces certain latency [29]. The technological risk of SOA can be challenging due to factors like early adoption and evolution of supporting technology, distributed infrastructure which requires high availability and scalability,
organizational change since SOA crosses system boundaries, efficiency in development and reuse, entity aggregation e.t.c. Also, new competences must be developed spanning project management, development and operations, analysis and design. Successfully solving problems and providing useful enterprise applications requires a combination of business, technology, architecture, organization, people and process.

1.1.3 SOA Life Cycle

Service-oriented architecture development is a gradual process that has many processing stages that can be referred to as life cycles. These gateways serve as a compass to follow to achieve an efficient and robust system. The SOA lifecycle is a representation that aims to demonstrate the association and dependencies between various independent processes that is made up of a mature, enterprise SOA program [117]. This includes the conceptualization and initiation stage, planning, development, deployment and continuous support even after implementation. The following constitute the basic stages in the life cycle of a typical service-oriented architecture development.

- **Development Stage**

  Mostly, organizations design and build services that match up to precise steps within a business process during the development phase. These services can be combined to produce a composite service or application for implementing specific business functions. The choice of which service interface to use to make available the services to the organization is then made for example; it can be through the use of Web services interfaces or some others. Since requirements are expected to change
overtime, also at this stage preparation for further development after the initial 
service is deployed is made and the process of managing the changes and cost 
implications are projected. A good service metadata management and service 
versioning enables organizations to enhance services and manage the deployment of 
multiple service versions in a cost effective and productive manner.

- **Integration Stage**

  After the service has been designed and the interface has also been developed, the 
  next step is to integrate it with other services or IT systems such as databases, 
  applications and transactional management systems since it is not going to be 
  working in isolation. These integrations mostly demands transformation of data to 
  map between diverse data schemas, as well as dynamic routing for linking the 
  appropriate services at run-time.

- **Orchestration Stage**

  After a couple of services have been developed, they can then be combined 
together step by step to create seamless, reliable process flows. The process of 
“gluing” services together with flow logic is called orchestration [117].

- **Securing Stage**

  It is very important that accesses to services are secured before they are deployed 
in any form. For example the processes for authorizing and authenticating users, as 
well as provisioning them and managing their identities, must be designed before 
sensitive information is exposed as a Web service.
• **Management Stage**

The management stage describes the definition and enforcement of service level agreements for services, and various operational policies like auditing and billing for service usage. Good management policies can guarantee an organization that their services will be most likely reliable, available and constantly monitored for exceptions or failures.

• **Accessing Stage**

Services can be accessed in different ways and are typically exposed to users through a portal or a composite Web application. It can also be accessed through wireless devices such as cell phones and handheld devices. An SOA environment supports multi-channel access to services which enables organizations to adapt user interfaces without modifying the underlying services. This provides the user an increased flexible to access those services.

• **Analysis Stage**

For operational administrators and workers to effectively monitor, analyze and respond to time-sensitive issues, the analysis of services, events and business processes involved in business operations often needs to occur in real time. This enables organizations to figure out difficulties encountered in their processes and inform the concerned personnel when a particular event warrants attention. Following a pattern as such is an approach that ensures proper security, reliability and availability of services.
1.1.4 SOAGovernance Policy

Service-oriented architecture governance is a major component among the best practices that describes how organization and process tie together the other components of the success equation e.g. architecture, business, and technology. Though the focus of services is not on core administrative functioning but rather on the development of a service that will be consumed by those both internal and external to the firm, without an effective SOA governance policy, enterprises will struggle to achieve the results that they desire. Governance processes must be adaptive and flexible in what the enterprise needs, what systems the enterprise will build, and how those systems will be built will be much different in the future compared to now [4]. Organizations must have discipline and rigor in the enforcement of the architectures, standards, and policies they adopt for SOA. Effective SOA governance should achieve four main goals:

- The deployed services are aligned with the business
- The services enable the business to achieve the benefits desired
- The services are delivered effectively
- The services are owned and orchestrated across the enterprise

Focusing on the above goals, a firm can deploy a governance framework that includes; service ownership, service orchestration, service alignment, service delivery, and service value [4]. These SOA governance areas are the issues that IT executives should focus on as they seek to impact the business with this flexible, agile technology and concentrate on the convergence of three aspects: people, processes, and technology. People are those who have the decision rights to make the necessary types of decisions, processes are how the decisions are made and what
mechanisms are used to determine if the goals were achieved. Technology is the facilitating mechanism used to facilitate the people and the processes within these SOA governance elements to make the decisions.

- **Service Ownership**

  Service ownership identifies the issue of who is responsible for the development, implementation, maintenance, and enhancement to the service. It is recommended that ownership be clearly delineated and shared between a business owner and an IT owner, and that the nature of the ownership should be negotiated and formalized to ensure service success.

- **Service Orchestration**

  Service orchestration refers to an enterprise-wide governance arrangement for examining proposed services to ensure that the needs of the enterprise are effectively being achieved. It prevents duplication of efforts and promotes the reuse of components in an efficient manner. Orchestration can be achieved by cross-functional review boards, an executive advisory board, or a mechanism that focuses on the enterprise rather than the individual business units.

- **Service Alignment**

  Service alignment focuses on ensuring simultaneous linkage between the services, business processes and the strategic IT plan. The services must fit within the broader IT framework by the organization to facilitate more effective business processes which require a governance mechanism to ensure that this occurs efficiently.
• **Service Delivery**

Service delivery refers to the governance arrangements designed to facilitate the distribution of services, including the base underlying architecture and infrastructure to ensure success. The execution of the services requires a governance arrangement that ensures reliability and consistency, and also requires that there is a high degree of commitment to delivery success. The delivery is ensured through service-level agreements (SLAs) that are co-developed by the business and IT.

• **Service Value**

Service value is about executing value propositions throughout the service lifecycle, ensuring that IT delivers the value that was originally proposed. This can be accomplished through the use of dashboards and metrics that should be negotiated, and focused on delivering value not only from the cost perspective but from a variety of other perspectives, captured by a balance scorecard.

1.2 **Web Service**

In this section the goal is to introduce Web services - the most important technology used in this thesis. It starts to give an overview introduction of Web services that gives the reader an illustration of technology history, advantages and architecture. The following sections will describe the selected Web services components and protocols. After this chapter the reader will have a general and clear image of Web services technology.
1.2 Introduction of Web Services

Service-oriented architecture can be implemented using various technologies like Web Services, Service Component Architecture (SCA), Enterprise JavaBeans (EJB), CORBA and so on. It is possible for a service developed to have different kinds of interfaces. For example, a service can have a web service interface and a Java-based SCA-service interface. However, Web services is the most common new technology for implementing Service oriented Architecture. Despite some current limitations, an SOA with Web services is an ideal combination of architecture and technology for consistently delivering robust, reusable services that support present business needs and that can without difficulty be adapted to satisfy changing business requirements.

In mid to late 2000 Web Services technology was introduced with the first version XML massaging – SOAP, WSDL 1.1, and a service register policy that was the initial version of UDDI. Those standards build a wide accepted interoperability among software components. Companies like IBM and Microsoft have been provided their Web services implementation products widely used to solve business problems.

The World Wide Web Consortium (W3C) is running a Web Services Activity and its goals to develop a set of technologies in order to lead Web Services to their full potential. The W3C defines Web Services as follows:

“Web services provide a standard means of inter-operating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well
as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.” (W3C 2002)[29]

The most valuable characteristics of Web services is using loosely coupled, standards-based technologies, and those two aspects are also the key concepts of Service-Oriented Architectures (SOA). The SOA is a methodology for achieving application interoperability and reusability with the following features:

“A strong architectural focus, including governance, process, modeling, and tools. An ideal level of abstraction for aligning business needs and technical capabilities, and creating reusable, coarse-grain business functionality. A deployment infrastructure on which new applications can quickly and easily be built. A reusable library of services for common business and IT functions.” [29]

The power and flexibility of SOAs can provide a services model that you can create new services base on improve existing ones without leaving the services paradigm. It means that if there is an organization gives its coarse-grained services and presents them in a clear functionality, the consumers could dynamically discover and bind to the available services in a flexible way - to build the application by composed services. The IT infrastructure behind those services can be flexible and reusable.

First, there need to be services which are presented with abstract definitions, including the detail information that allows anybody who wants to use those services to and bind it. The ‘Requestor’ bind the specific ‘Services’ and acquire its functions.
Second, the services providers who want to let people find their services need to publish details of their service to a facility. The details include precise description how people could obtain those services and business information. Third, the published services need a discovery facility to let people find those services. This illustration is the notion of a infrastructure that supports SOA, and the goal of Web services technology is to address these questions.

“SoA represents an abstract architectural concept. It is an approach to build software systems that is based on loosely coupled components (services) that have been described in a uniform way and that can be discovered and composed. Web services represent one important approach to realizing a SOA.” [119]

The different between Web services approach and traditional approaches first the loose coupling aspect of the architecture. Traditional approaches build applications where the objects or component are tightly related to each other by functionality requirement. But Web services do it in more dynamic and adaptable way based on well known and understood services; second though Web services themselves are developed in an open way, organizations such as W3C and the Organization for the Advancement of Structured Information Standards (OASIS) provide Web services in standards and technologies that are the foundation of the Internet.

The classic distributed system has a very important concept – components, the components goals the reuse of tested partial solutions and easy system integration. This concept reduce the cost for develop a new system, and that the new system has nice competition ability. But current distributed system has issues: typically, current distributed system technology is based on object systems. In that
case, a service is similar to a method of a class implemented by an object. When people want uses a single method of the service it needs to use the whole class. The requestor and service are tightly coupled, when the service class hierarchy changes the requestor must change the application that uses that class. The interoperability is also a problem for current distributed system. “Different distributed system technologies such as Common Object Request Broker Architecture (CORBA), Java 2 Platform, Enterprise Edition (J2EE) and Component Object Model (COM) are base on quite different and incompatible object models [89]. The interoperability between those platforms is difficult.

A few years ago, businesses finding themselves in need of comprehensive integration solutions turned to products and practices developed specifically for that purpose. However, these enterprise application integration (EAI) products proved to be expensive, consumed considerable time and effort, and were subject to high project failure rates. Furthermore, because these various special purpose products are proprietary, many of the projects resulted in additional difficulties whenever a company invested in more than one of them.

Recent experience shows that a better answer is available by using Web services standards. Instead of dealing with the complexity of multiple incompatible applications on multiple computers, programming languages, and application packages by introducing an EAI product, it’s possible to add a layer of abstraction that’s open, standards-based, and easy to integrate with virtually any new and existing environment.

According to the Gartner 2008 hype cycle for emerging technologies [119] basic Web services are seen to be fairly mature and reaching the "plateau of productivity,"
while SOA is placed on the "slope of enlightenment" as costs and benefits are now being viewed more realistically. Best practices for SOA, however, have still yet to mature [33]. SOA adoption overall is increasing, albeit slower than initially anticipated [42]. Consequently, the question of how to arrive at an SOA that enables the expected business benefits at an acceptable cost merits further examination and best practices need to be developed.

The combination of Web services and SOA provides a rapid integration solution that more quickly and easily aligns IT investments and corporate strategies by focusing on shared data and reusable services rather than proprietary integration products.

It is possible to reuse a traditional three-tier application as a service-oriented application by creating services at the business logic layer and integrating that application with other applications using the service bus. Another benefit of service orientation is that it’s easier to separate the presentation logic from the business logic when the business logic layer is service-enabled. It’s easier to connect various types of GUIs and mobile devices to the application when the business logic layer is service-enabled than if a separate tightly coupled presentation logic layer has to be written for each. Instead of running the presentation logic tier as a tightly coupled interface on the same server, the presentation logic can be hosted on a separate device, and communication with the application can be performed using the service bus [59].

Applications can more easily exchange data by using a Web service defined at the business logic layer than by using a different integration technology because Web services represent a common standard across all types of software. XML can be used
to independently define the data types and structures. And finally, the development of service-oriented entry points at the business logic tier allows a business process management engine to drive an automatic flow of execution across the multiple services [59].

Creating a common Web services layer or “overlay” of services into the business logic tiers of applications also allows you to use a common service repository in which to store and retrieve service descriptions. If a new application wishes to use an existing service into one of these applications, it can query the repository to obtain the service description and easily generate SOAP messages to interact with it.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [33]. It is described using a standard, formal data centric XML notation, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and locations. The most common service descriptions are:

• The Simple Object Access Protocol (SOAP) [29] that is the fundamental messaging framework of web services. This protocol provides extendable XML-based messaging allowing data exchange between distributed applications and may be used with different network transport protocols (e.g., Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP)).

• Web Service Description Language (WSDL) [59] is an XML-based service description on how to communicate and make a request to a Web service. WSDL is
often used in combination with SOAP and XML Schema to provide web services over the internet.

Universal Description Discovery and Integration (UDDI) [60] is the protocol used for the publishing of the Web services information in order to find available Web services on the Internet or an intranet. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the Web services listed in its directory.

WSDL description is used to get the syntactic information about the definition of each service provided by a government agency. Each service must have a WSDL document containing an XML description of the functional aspects of the service, as well as the input and output parameters for its operations. Communication between service requester (citizen interface) and service provider (agency) is carried through SOAP messages.

A web service provides an access interface to a service. This interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which this is implemented and of the programming language in which this is written.

1.2.2 Web Services Description Language (WSDL)

Each web service has associated a WSDL document which contains information about what messages go in and come out from a service. This information is only syntactic or structural but not semantic information.
From the point of view of a client a WSDL document describes a public service for it, and from service implementer side, the document can be viewed as the documentation of a standard service to be used.

A WSDL document has two parts: an abstract part and a concert part [59]. The fistone, describes what Web service does in terms of messages it consumes and produces, and the other one, defines how and where the service is offered.

Figure 2. Structure of WSDL

Figure 2 shows the syntactic structure of a WSDL document sticking out the abstract part and concrete part. As mentioned documentation provided by a WSDL file is XML-based. It is formed by a set of definitions contained into a <definitions> element. This element contains a target name attribute who is the ownership of the set of definitions inside the document, all the XML namespace needed (xmlns attributes) and an optional name attribute related with the name of the Web Service. A <definitions> element is composed by the following elements:

- <type> element: It is used to indicate the data definitions that web service will use later in the definition of the messages that it will exchange. WSDL does not declare data structures per se. It allows languages that can define XML data to be used.
• `<message>` element: It allows describing the messages that web service will exchange. One `<message>` element must be defined by each message that the web service exchanges. A message is composed by one or more parts where a `part` is associated conceptually with a parameter; its structure depends on how the binding area of the WSDL is written. Each `part` has two attributes, its name and the kind (element or type) which is associated with the messaging style. If a part defines `type`, it means it is using the RPC-style messaging and it can be associated with any data item of the specified type, otherwise, if the part defines `element`, it means the document-style messaging is used and the part must be the value of a specified element.

• `<portType>` element: It contains one attribute, the port Type `name`, it is only an identifier. This element is probably the most important part of a WSDL document essentially because it structures the set of `<operation>`s that a Web service provides and the group of inputs, outputs and faults message that it will be required or produce. Generally, there is only one `<portType>` element by WSDL document but it is not restrictive.

The message exchange defined by a `<operation>` can be:

- **One-way**: Service receives one message but it has not to produce anything message in response.
- **Request-response**: Service receives one message and it has to produce a message in response.
- **Solicit-response**: Service sends a message and gets a response back.
- **Notification**: Service sends a message and it does not receive anything message in response.
• **<binding>** element: While **<portType>** specify what **<operation>s** can be carried out and the **<messages>s** define the parameters types for the operations, the **<binding>** elements describe how to format those messages to interact in a protocol-depended manner. WSDL does not assume one standard way to format messages; it can exchange messages using SOAP, HTTP or MIME, but the most common is SOAP binding.

• **<service>** element: It is the final part of a web service description which contains one attribute, its **name**, and a child element **<port>** which has two attributes, name and a reference to a binding. A service can have several **<port>** elements. Each **<port>** element describes where a single portType is offered via a given binding.

### 1.2.3 XML Schemas

XML provides a set of structures that can represent many different types of document and data-oriented information [33]. When there is a necessity of establish a contract among multiple parts which are working with the same type of XML, it is essential to define specific rules that the XML document of each part must comply. After a bigger effort the W3C produced a standard specification to established, XML Schemas. An XML Schema Definition or XSD is an XML-based document that describes the structure of an XML file.

There are many advantages of XML in a broad range of areas. Some of the factors that influenced the wide acceptance of XML are:

Acceptability of use for data transfer: XML is a standard way of putting information in a format that can be processed and exchanged across different hardware devices, operating systems, software applications, and the Web.

Uniformity
and conformity: XML gives a common format that could be developed upon and is accepted industry-wide. Simplicity and openness: Information coded in XML is human readable.

### 1.2.4 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a network, transport, and programming language neutral protocol that allow a service consumer to call a remote service provider. The message format is XML. The currently adopted standard is W3C’s SOAP 1.1 specification.

SOAP has the following characteristics:

- SOAP is designed to be simple and extensible.

  SOAP provides a framework to describe message content and process instructions, and an optional set of encoding rules for representing defined data types.

- All SOAP messages are encoded using XML.

  SOAP is transport protocol independent. HTTP is one of the supported transports. Hence, SOAP can be run over an existing Internet infrastructure.

  There is no distributed garbage collection. Therefore, call by reference is not supported by SOAP; a SOAP client does not hold any stateful references to remote objects.

  SOAP is operating system independent and not tied to any programming language or component technology. It is object model neutral.
Given the Web’s intrinsically distributed and heterogeneous nature, communication mechanisms must be platform-independent, international, secure, and as lightweight as possible. XML is now firmly established as the lingua franca for information and data encoding for platform independence and internationalization. Building a communication protocol using XML is thus a natural answer for Web services. SOAP works on existing transports, such as HTTP, SMTP, and Web sphere MQ. At its core, a SOAP message has a very simple structure: an XML element with two child elements, one of which contains the header and the other the body. The header contents and body elements are themselves arbitrary XML. In addition to the basic message structure, the SOAP specification defines a model that dictates how recipients should process SOAP messages. The message model also includes actors, which indicate who should process the message. A message can identify actors that indicate a series of intermediaries that process the message parts meant for them and pass on the rest [29].

1.2.5 Universal Description Discovery and Integration (UDDI)

The transport, description, and message layer are the fundamental of Web services, its means with those three layers people could build the minimal Web services platform to communicate in an interoperable way using messages. But in SOA architecture, discovery services and negotiation are very important aspects, it need a way to provide the features for discovery of services and the descriptions about the agreement between a requester and a service. In this section, the most used Web services technologies to solve this problem is introduced.

The Universal Description, Discovery, and Integration (UDDI) is an important Web services aspect which is widely accepted and used. It provides a
solution for users to find required services from a well-known facility or registry their own services. In this scenario, the services metadata could publish in a form that is discoverable and searchable by users who are looking for appropriate services they require to solve their particular problem. Also, the organizations might be able to publish their services by register the metadata describing the interfaces to their services, and enable domain-specific taxonomies of services.

“UDDI began as collaboration among Microsoft, IBM, and Ariba to promote the adoption and use of Web services standards.” (Newcomer 2002, p.153) [95] Though the rapid development, those companies founded UDDI.org and invited more and more groups and companies to participate. In July 2002 the UDDI.org was absorbed by OASIS and SAP replaced Ariba as a registry host site. UDDI evolution goes from v1 to v3. Currently UDDI v3 has been widely acknowledged and used. Compared to v2 it is a pretty improvement and with specifically improved security.

UDDI has two main parts: registration and discovery. In registration part, the businesses can post their services information to UDDI repository so that other businesses can search for and discover them, which is the discovery part. The businesses and individual users could interact with UDDI by using SOAP APIs or the user interfaces provided by the operator or other Web services vendors, for example the developers can find service from companies own web site or its own UDDI repository. Normally, UDDI repositories could be provided in one of three ways:

“Public UDDI – These are UDDI repositories that can serve as a resource for Internet-based Web services. An example of this is the UDDI Business Registry [UBR] – hosted by a group of vendors led by IBM, Microsoft, and SAP – that is replicated across multiple hosting organizations.
Intra Enterprise UDDI – An enterprise has a private internal UDDI repository that provides much more control over which service descriptions are allowed to be placed there and used by application developers within that specific enterprise.

Inter Enterprise UDDI – This basically scopes the content of the UDDI to services that are shareable between specific business partners.” [89]

Comparing UDDI in real world model, it is more like a yellow page phone book. People could find the information about who offer the services; information about the particular family and technical offering; and the information about services access endpoint and integration specification. The UDDI information is often described individually so people could search and query in a nice way.

1.3 Legacy System

Legacy systems are often at the center of business procedures of large organizations resulting in a need to modernize these systems as they can begin to resist change over time. There are many different descriptions for legacy systems. Legacy systems are large software systems that people do not how to deal with, but are important to the organization. A legacy system could be any information system that significantly resists modification and evolution to meet new and constantly changing business requirements [17]. It is critical software that cannot be modified efficiently. It is a system that was developed at some time in the past and which is critical to the business in which the system operates. Many legacy systems remain supportive of core business functions and are ‘indispensable” to the business [62].
From the above definitions, it is clear that legacy systems are of great value and essential for a successful organization. Hence the same has to be maintained and has to be reused based on recent integration technology world.

Legacy systems have two aspects namely static and dynamic. Static aspects include legacy assets such as data and applications. The static modeling, represents subsystems, or logically connected software elements. The dynamic aspects include the events that perform tasks. In dynamic modeling behavioral descriptions that show interactions amongst high-level elements are described.

Generally legacy systems are developed using COBOL, Assembler, C, etc., Along with the development of technology legacy systems have been expanded to contain Java, XML, C++ etc.,

1.3.1 Virtual Sequential Access Method (VSAM)

VSAM is one of several access methods in z/OS. It only applies to data stored in DASD devices. This access method makes it easier for an application to execute an I/O operation (moving data between an I/O device and memory). There are three types of access in VSAM:

- Random: This is also referred to as direct access. The logical record needs to be located by the use of a search argument coming from the application. There is no search argument connection between two consecutive logical record accesses.

- Sequential: The entire file is processed (for Read or Write), one logical record after the other. The application does not need to provide any search argument.
The access method can implement a Read look ahead technique to load logical records in the buffers not yet required by the application program.

- **Skip sequential**: A combination of the two previous types of access. The application randomly provides one search argument and from the located logical record on, all records are processed sequentially. An example is sequentially processing all the customers of a bank branch office in a file that has all the customers of the bank.

### 1.3.1.1 VSAM data set organizations

VSAM arranges records by index key, relative record number, or relative byte address. It is used for direct or sequential processing of fixed-length and variable length records on DASD[23]. VSAM data is cataloged for easy retrieval. VSAM supports five data set organizations:

- **Key-sequenced data set (KSDS)**
- **Entry-sequenced data set (ESDS)**
- **Relative record data set (RRDS)**, of which there are two types:
  - **Fixed-length relative record data set (RRDS)**
  - **Variable-length relative record data set (VRRDS)**
- **Linear data set (LDS)**
- **HFS files**
1.3.2 Database (DB2)

DB2 is a formal subsystem of the z/OS and OS/390 operating system. Understanding the structure of DB2 can help conceptualize end-to-end data flow. An address space is a complete range of storage addresses bound by a beginning and an ending address that can be accessed by the operating system. Although there are different types of address spaces, each address space can be considered its own entity, which operates independently. In other words, an address space can be thought of as a logical box of memory. DB2 for z/OS and OS/390 uses a number of address spaces.

1.3.2.1 DB2 packages, plans, collections, and package lists

An application that contains embedded SQL is precompiled to extract the SQL statements from the source code. This is done because the source code language compiler does not recognize SQL statements as legitimate syntax of the source code language used. Therefore, the source code must be run through the DB2 precompiler before being compiled by the source code compiler. Alternatively, if the source code compiler provides a SQL statement coprocessor, there is no need to run the source code through the DB2 precompiler. With the SQL statement coprocessor, the SQL statements are extracted and the source code is compiled in one step.

- Packages and plans

A DBRM contains a modified rendition of all the SQL statements extracted from the source program. The DBRM communicates the SQL requests to DB2 during the bind process. With the DBRM generated by the DB2 precompiler or the SQL statement processor, it can be either bound into a package or directly into an
application plan. Each package contains only one DBRM. When a package is bound, there are many bind options that can be specified. These options determine how the SQL statements in the DBRM will ultimately execute in DB2. Bind options are very powerful.

One or more DBRMs can be directly bound in a plan. One or more packages can be bound in a plan. A plan can be made up of both DBRMs and packages. In almost all cases, an application must have a plan bound before it can run. There is one exception to this rule. A (explicit) plan is not required when using a distributed connection into DB2.

1.3.3 Customer Information Control System (CICS)

Mainframes as a single system can scale up to process a billion transactions per day. Mainframes support different kinds of workloads, which can be defined in two categories, “basic” batch processing often old applications running during nights, to make statistics and long jobs, and online transaction processing, which are the most used during days.

**Batch processing** doesn’t require any user interaction. They’re often planned to be executed nightly, when all machine power is available. They have to advantages to be able to process huge data, such a as terabytes to create valuable statistics.

**Online processing** occurs interactively with users. Unlike batches processing, they have to be executed very fast, and response time is the most important thing with, of course, data integrity. All online transactions often depend of the core business functions which are so critical and have to be handled with utmost
care. Online processing requires immediate response, which provides high performance, integrity, and data protection.

Mainframe system achieves the robust online processing using a subsystem called as Customer Information Control System (CICS). Large number of industry used mainframe because of the input and output capacity. Moreover Mainframe had all its client, server and database layers all situated in same hardware. Data check and processing is far more speed than any other system. As a result, IBM designed its machine as “balanced systems”. It means it balances server’s components to processor, memory and I/O scalability. It’s then able to deal with large quantities of data available to support transactions [34].

CICS (Customer Information Control System) is an online transaction processing system which controls information by providing system management, database and network communication functions. It provides an interface between application programs and operating system. It runs as a unique z/OS batch job and allows hundreds of users to interactively access several applications.

CICS is one of the often used application platforms [33] because of its proven Quality of service factors such as performance, scalability, Reliability and Availability. CICS application can be developed by various programming approaches such as Conversational approach, Non-conversational approach and pseudo – conversational approach.

In Conversational approach, the application makes use of the resources even during the think time of execution cycle, which leads to deadlock situation if think
time is high. Hence the resource efficiency is less. Non-conversational approaches are not suitable for interactive online applications. Hence this approach is not valid.

CICS applications are always pseudo conversational in nature. According to user it appears to be interactive application, but the resources are released to the system during the think time. Hence the resources are efficiently used.

CICS Support programming language Java, C/C++, Enterprise COBOL, Assembler, PL/1, REXX. Object oriented programming support, even in Assembler, with IBM's High Level Assembler.

The CICS Application Programming Interface helps isolate the CICS application programmer from the operating system, allowing application programmers to focus on application development and business solutions [33].

CICS offers [14] flexible intercommunication facilities allowing it to be supported and configured across a variety of networks. CICS data management is comprehensive and includes support for major database systems such as DB2, Oracle, IMS – Hierarchical database, etc.

A recent enhancement to CICS is the ability to participate in Service Oriented Architecture. CICS is strategically positioned for SOA and Web Services computing.

Challenges of CICS Transaction

- Are inflexible and not serviceable
- Are redundantly implemented
- No modularization, all presentation logic, business logic, and database logic and are not clearly capsulated.
• Limited Graphical User Interface (GUI) elements for user interaction
• Impossible to invoke a sequence of CICS transactions automatically.

Today's business highly depends on number of customers. To manage and maintain them better customer services should be provided for customer. This is one of the challenging factors for legacy system.

• The cost involved in maintaining legacy application is much higher. This is also one of the primary factors for modernizing legacy application. The cost of MIPS in mainframe platform is much higher than MIPS in non mainframe platform.

• The mainframe skill sets are also drastically decreasing. Hence it is difficult to maintain and operate existing legacy application.

• Today there is an option to use advanced IDE, testing and debugging tools. Hence the productivity would improve. But many of the business critical mainframe applications are running without proper IDE support.

• The monolithic application architecture of mainframe system contains business logic, presentation logic and data access logic together that is the main reason for lack of flexibility. Hence it affects the reusability and availability of an application [3].

1.3.4 Business Rule

A business rule is anything that imposes structure upon or controls the behavior of a business practice. A rule can enforce business policy, establish common guidelines within an organization, or control access in a business environment.
More commonly, business rules are discovered and documented informally during the initial stages of a project. In this case the collecting of the business rules is coincidental. In addition, business projects, such as the launching of a new product or the re-engineering of a complex process, might lead to the definition of new business rules. This practice of coincidental business rule gathering is vulnerable to the creation of inconsistent or even conflicting business rules within different organizational units, or within the same organizational unit over time. This inconsistency creates problems that can be difficult to find and fix.

Allowing business rules to be documented during the course of business projects is less expensive and easier to accomplish than the first approach, but if the rules are not collected in a consistent manner, they are not valuable. In order to teach business people about the best ways to gather and document business rules, experts in business analysis have created the Business Rules Methodology. This methodology defines a process of capturing business rules in natural language, in a verifiable and understandable way. This process is not difficult to learn, can be performed in real-time, and empowers business stakeholders to manage their own business rules in a consistent manner.

Gathering business rules is also called rules harvesting or business rule mining. The business analyst or consultant can extract the rules from IT documentation (like use cases, specifications or system code). They may also organize workshops and interviews with subject matter experts (commonly abbreviated as SMEs). Software technologies designed to capture business rules through analysis of legacy source code or of actual user behavior can accelerate the rule gathering processing.
1.3.4.1 Categories of business rules

According to the white paper by the Business Rules Group, a statement of a business rule falls into one of four categories:

- Definitions of business terms

The most basic element of a business rule is the language used to express it. The very definition of a term is itself a business rule that describes how people think and talk about things. Thus, defining a term is establishing a category of business rule. Terms have traditionally been documented in a Glossary or as entities in a conceptual model.

- Facts relating terms to each other

The nature or operating structure of an organization can be described in terms of the facts that relate terms to each other. To say that a customer can place an order is a business rule. Facts can be documented as natural language sentences or as relationships, attributes, and generalization structures in a graphical model.

- Constraints (also called "action assertions")

Every enterprise constrains behavior in some way, and this is closely related to constraints on what data may or may not be updated. To prevent a record from being made is, in many cases, to prevent an action from taking place.

- Derivations

Business rules (including laws of nature) define how knowledge in one form may be transformed into other knowledge, possibly in a different form.
1.3.5 Legacy modernization Approaches

Legacy Modernization categories can be identified by existing application analysis. The first phase of modernization is asset analysis and this is important to finalize on the modernization categories.

Usually, most legacy software systems face with some problems such as dependency on hardware, lack of document, impossibility of tracing the program, lack of integrity and unity of system, increasing the time and supportive costs. These problems cause that organizations perform several activities for modernizing their legacy information based systems. To overcome these problems, system developers, use one of these approaches

- Reengineering
- Migration
- Replacement
- Integration

1.3.5.1 Reengineering

In evolution of Information system, the system is developed based on the specification and requirement at a given period of time. Later when the time grows, the functionality of the system also grows and it becomes necessary that the system has to be maintained to accommodate the requirement or functionality. Later point of time maintaining the system for a longer time may degrade the performance of the system.
The importance of a legacy system is information flow and act as main drive for consolidating business information is widely recognized by both scientific and industrial communities. These systems and the data they process are vital assets for the organizations that use them. The organization’s evolution through the years requires synchronized evolution of the legacy systems; however, such systems should always provide an adequate quality level, so that they can be easily maintained. Unfortunately, due to degradation, legacy systems very often provide low quality levels, and, as a consequence, their maintenance becomes very costly.

Reengineering approach includes a combination of reverse engineering process, re-documentation of existing system, re-structuring the existing system, forward engineering on new documents for the purpose of developing new system and replacing new target system with existing system. The aim of this approach is understand available legacy system including characteristics, designs and implementations and re-carry the system for improving its efficiency.

Lehman empirically proves in [71] that if no improvement is made, maintenance activity degrades the software quality and, therefore, its performance. In [105] some quality factors, called aging symptoms, are identified; each of them is then associated to a set of metrics, which allows its quantification. These symptoms become heavier and heavier to manage as the number of maintenance activities increases, so confirming the principles expressed in [71]. Moreover, in [110] some experimental evidence was derived, which showed that the reengineering process can decrease the performance and some functionality can be achieved because of some aging symptoms.
The benefits of Reengineering approaches are

- Automatically preserves all business functionality.
- Target system has all business needs being addressed with enhancement of existing system with new functionalities.
- Target system has a modern technical architecture with the flexibility to adapt to future changes in the system.

Limitations of reengineering approach

- The reengineering process is complex because it becomes necessary that the data and the procedures to be restructured all at the same time.
- Moreover, according to other authors such as Biggerstaff [102] and Brown [5.6], the reengineering process must involve the entire system. It becomes necessary that the system has to be blocked during execution of the process and also the maintenance activities should be interrupted until the process is concluded.
- In addition to this each change would have to be executed parallel both in the legacy and in the reengineered system, and there is a high risk that the new system would no longer be equivalent to the legacy at the end of the process, therefore further corrective maintenance would be required. This situation causes a loop between the maintenance process and the reengineering process.

1.3.5.2 Migration

Most legacy applications were custom-developed and hard-wired to support proprietary business logic and exploit the capabilities of the hardware and software
platforms for which they were written, making them both efficient and reliable. These applications are highly optimized and it is not therefore surprising; that even with the advent of new technologies.

However the same characteristics that make legacy applications the mainstays of business computing have also made them difficult to integrate with one another and especially with the newer and evolving technologies. These legacy applications are self-contained and have little or no flexibility to adapt them to changing business requirements. However, some of our current business models were not even conceived of at the time these applications were developed. In general, legacy application maintenance is very cumbersome and represents a major fixed cost at a time of reduced IT spending.

Data in legacy systems is an important business resource. It represents mission-critical business knowledge which cannot be easily replaced. The only solution to make use of the data might be legacy migration. There are lots of migration approaches such as system, data, code and architectural migrations.

Many systems have successfully implemented the data and code migration to newer platforms. In view of this, many organizations were ready to migrate their legacy systems to newer technologies.

Legacy system migration encompasses many research areas. A single migration project could, quite legitimately, address the areas of reverse engineering, business reengineering, schema mapping and translation, data transformation, application development, human computer-interaction and testing. Due to space
limitations, only a brief outline of research directly related to legacy system migration will be presented.

In this method the legacy and target information systems are operated in parallel throughout the migration. Initially the target information system is very small, perhaps only one application with a very small database. However as the migration progresses the target system will grow in size until it performs all the functionality of the legacy system which can then be retired. During the migration, the legacy and target information systems interoperate to form the operational mission-critical information system. This interoperability is provided by a module known, in general, as a gateway, “a software module introduced between operation software components to mediate between them” [11].

Benefits of Migration

- The data migration methodologies can provide interoperability between the legacy system and its target system to access data.
- Application migration and data conversion to new technologies also creates opportunities for restructuring to improve accuracy and maintainability.
- Migration to consolidated platforms can also improve performance.

Limitations of Migration

- In addition, they also led to some difficult problems, such as maintaining the consistency of the data between the two systems.
So the final solution for this might be the third factor mentioned above, that is modernizing application connectivity, existing application components can be reused and new solutions can be integrated.

1.3.5.3 Integration

Integration is getting all IT resources while taking advantage of next generation technologies. An integration strategy can be intrusive or non-intrusive. An integration methodology requires knowledge of the internal working of a legacy system is called white box integration, while integration methodology that requires the knowledge of external interfaces of a legacy system is called black box integration [5.10]. If application source code is modified, the connection is considered intrusive. Intrusive connections are used when custom coding is developed to handle specific application needs or to increase performance. Non-intrusive connections are used if the information required from the application is already available from an existing interface and the transaction volume is low to moderate. There are two major approaches for legacy systems integration, application integration and data integration.

In Application integration, the core business logic is already available and this application has to be persevered by extending the interface to the application to interoperate with distributed applications.

In Data migration, the data and metadata can be easily manipulated directly by applications in the new architecture by integration with distributed applications.

The modernization solution has to be identified by analyzing the recommended solution with various factors like cost, effort, risk involved etc.,
<table>
<thead>
<tr>
<th></th>
<th>Integration</th>
<th>Reengineering</th>
<th>Migration</th>
<th>Replace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Effort</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Reuse of legacy</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Nil</td>
</tr>
<tr>
<td>Risk</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Functionalities</td>
<td>Same as Legacy system</td>
<td>Same as legacy with added functionalities</td>
<td>Same as legacy with some added functionalities</td>
<td>Completely</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Same as Legacy system</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Change is core business logic</td>
<td>Nil</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Legacy Contribution</td>
<td>High</td>
<td>Nil</td>
<td>Nil</td>
<td>Nil</td>
</tr>
<tr>
<td>Skill set</td>
<td>Newer and Same as Legacy system</td>
<td>Newer</td>
<td>Newer</td>
<td>Newer</td>
</tr>
</tbody>
</table>

*Table 1 Comparative Analysis for Legacy modernization approaches*

From the above table it is clear that based on various factors integration is the ideal solution to overcome the challenges of CICS.

1.4 Need for the Study

A legacy system can be a computer system or application program that is still used because of the prohibitive cost of replacing or redesigning it, despite its poor competitiveness and compatibility with modern counterparts [79]. They are
invaluable assets with embedded business logic representing many years of coding, developments, modifications and enhancements. Mostly these systems were developed independently without a consistent underlying architecture, resulting in overlapping and redundant functionality and data. Therefore a better approach is to find a way of modernizing these systems to remain competitive. However, modernized IT solutions must create new value from existing systems and provide flexibility and easy interoperability among a broad set of technologies which is usually a problem with legacy applications. Service-oriented architecture offers a practical solution for evolving and reusing existing assets. Transforming a legacy system requires a deep functional knowledge of the system [79]. It is vital to understand the business processes that are automated by the legacy system and to compare those with the current needs with the aim of improving the outdated parts to create new value from existing assets. Since the legacy systems are based on outdated technological platforms that are often difficult to maintain and possibly not supported anymore, the target architecture must be based on leading-edge technologies, maximizing the reuse of existing assets.

Legacy systems typically form the backbone of the information flow within an organization and are the main vehicle for consolidating information about the business. If one of these systems stops working, the business may grind to a halt. These mission critical legacy information systems are currently posing numerous and important problems to their host organizations. In particular these systems usually run on obsolete hardware which is slow and expensive to maintain, maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal
workings of the system; integration with other systems is greatly hampered by the absence of clean interfaces;

Major changes in business practice inevitably require major changes to the supporting information systems. However, legacy systems are characterized as being very brittle with respect to change. Small modifications or enhancements can lead to unexpected system failures which are very difficult to trace in a largely undocumented environment. Many organizations now wish to move their legacy systems to new environments which allow information systems to be more easily maintained and adapted to new business requirements. The essence of Legacy Information System Migration is to allow them to do this, retaining the functionality of existing information systems without having to completely redevelop them. Of the currently available, and generally ad-hoc, methods for legacy system migration, few have had, limited, success [67], [80].

It is reported that over five billion lines of COBOL code are added annually on a base of over 200 billion lines of code. Legacy systems support core business processes and provide crucial information for day-to-day operations. They contain business logic that provides competitive differentiation and institutional knowledge in the form of customer, product, supply chain and channel partner data. They also are optimized for performance and scale and embody many of the competitive advantages that enterprises need to succeed. They are, however, costly to maintain, rigid and difficult to use in new ways without unpredictable or even negative results.

Legacy systems generally consist of invaluable assets with embedded business logic representing many years of coding, developments, enhancements, and modifications. However, they are often undocumented, tightly coupled, and relatively
closed and inflexible. In most cases they were developed independently without a consistent underlying architecture, resulting in overlapping and redundant functionality and data as shown in Figure 1.3.

![Diagram of Existing legacy Environment](image)

Figure 1.3. Existing legacy Environment

The main pain points of legacy assets usually fall into one of these issues:

- High cost of ownership, including costs of maintenance, operation, and upgrade of both software and hardware; for example, CPU usage-based pricing for mainframes.

- Slow time to market because of complex and poorly understood code. This may prevent the system from satisfying the evolving business requirements because simple changes take too long to complete and test. Changes tend to cause significant ripple effects and require more regression testing. This, in turn, increases maintenance and evolution costs.

- Monolithic architecture with little or no modularity together with redundant code. This is usually related to extensive patches and modifications as well as
duplicated and too-similar functionality implemented in different systems by separate teams.

- Closed and outdated technology that is difficult to integrate and interface with new, open technologies and modern distributed architectures.

- Shrinking talent pool of developers skilled in legacy systems and decreasing vendor support. Knowledge of these systems is usually restricted to a core set of people who are difficult to replace.

- Lack of application knowledge due to the departure of original developers or users, as well as missing or obsolete documentation.

Because most companies have a significant investment in their legacy infrastructure, management is typically not open to ripping out and replacing legacy systems, regardless of the level of shortcomings evident in the infrastructure. Rewriting or significantly modifying large portions of a legacy environment is neither practical nor realistically accomplishable in a reasonable time frame.

In order to enable legacy applications to participate in dynamic e-business, it becomes necessary to apply Service Oriented Architecture framework by implementing web service technologies which will allow the services to be defined so as to hide some of the complexity of the legacy application interface.

1.5 Motivation of the Research

The major problem in solving the pain points just mentioned is discovering and identifying the useful legacy functionality and tasks that can be exposed as services to be used by collaborating applications that seek to consume those services. Past architecture
designs typically considered legacy systems and black boxes. Standard application program interfaces (APIs) and interfaces were written to access legacy-based systems. Understanding the underlying business processes captured by the legacy systems were typically not addressed. In the black box approach, nonfunctional requirements (NFRs), such as performance, security, management, and maintainability, were often overlooked in the architectural design. Most often the applications and data running in the perceived black box were tightly coupled. This tight coupling makes it almost impossible to modify and reproduce the business processes.

In Service Oriented Architecture environment, business tasks are accomplished by executing a series of "services," which sometime participate in a business process. The services have a well-defined way of talking to other services and well-defined ways in which they talk back. The implementation of a service does not matter to a user as long as the service responds in the expected way and offers the quality of service he or she requires. This approach means that the service must be secure, reliable, and fast, and it makes SOA ideal for use in an IT environment where software and hardware from multiple vendors is deployed, or one in which existing IT assets are mixed with newer applications, integration technologies, or data sources.

Many business and IT benefits are realized from using SOA for legacy enablement. Foremost on the business side is that it creates new value from existing assets and systems, typically from new business processes and composite applications (for example, portal applications). SOA can help provide real-time access to what were previously batch transactions, thereby increasing the speed and accuracy of making business decisions. Reusing critical business data and applications through SOA can help deliver better customer service, thereby improving retention of those customers.
SOA allows to take advantage of superb quality of services while also repurposing critical processes and data. Furthermore, SOA can help you extend and protect your existing legacy investments and developer skills while helping you achieve greater interoperability with other systems in your enterprise, as well as those of customers, partners, and suppliers.

It is possible to get the best of the old and the new, to take advantage of technology progress while leveraging your existing assets. When you start doing this, you will be on your way to becoming a business that is more flexible and better able to respond to opportunities to better serve your customers and improve your operations. This agility is what is meant by "on demand business," and SOA can make your legacy infrastructure continue to work for you in new and better ways.

1.6 Objectives of the Research Work

This thesis aims at developing a framework that can be used by introducing service-oriented architecture through the use of Web services especially when migrating from legacy systems. Adopting any form of technology, architecture or services in organizations require a serious thought. For example, the requirements that must be fulfilled before adoption in terms of resources, the impact it will have on the organization both now and in the future among others. For the purpose of developing this framework, the thesis aims at working at following research areas

- To Modernize Legacy system through integration using service oriented architecture framework.
- To conduct a comparative analysis for legacy modernization using Migration, Reengineering and Integration using SOA.
• To Exploit a service-oriented architecture that will allow for dynamic coordinated and distributed building services management.

• To conduct research to identify and reuse critical business assets and leveraging them to be invoked in heterogeneous environment without affecting the performance and security.

• To conduct research to map data from a relational database based legacy application to Web Service SOAP messages suitable for Service Oriented Architecture Framework.

• To Enable VSAM data as service provider in SOA framework to be invoked in heterogeneous environment.

• To use Web service in SOA Integration framework to access Legacy online applications as a service provider.

• To use External services existing in distributed platform to be invoked by legacy batch application using SOA framework.

This will be achieved through research into relevant academic literatures, published articles, and journals in the use of Web services and SOA. A careful analysis of information will be carried out to compare the information presented from these sources and will further be developed to formulate the guideline that will be the result of this research study. This guide will offer tremendous information and assistance to organizations with the intention of introducing service oriented architecture and help them to avoid pitfalls encountered by early adopters through in depth understanding of some best practices and realize the full benefits of service oriented architecture. Organizations that have already introduced SOA can also benefit from this guide for further improvement and maintenance. There is no technology available today that thus not have its own weaknesses or areas that need to be improved upon. The results presented in this research is not sufficient to cover all the areas related to implementing service-oriented architecture using Web services, therefore further areas of study will be identified especially the weak areas where detailed study can be carried out for improvement.
1.7 Organization of Thesis

Chapter 1 introduces the broad area of Research topic such as Service Oriented Architecture, SOA life Cycle, Governance Policy. Web service and its standards such as WSDL, XML, and UDDI. It also gives an idea about Legacy system and its sub systems such as CICS, DB2, Business rules, VSAM and type of datasets such as KSDS, ESDS, RRDS and LDS. It also summarizes the benefits and limitations of legacy modernization approaches such as Reengineering, Migration and Integration.

Chapter 2 provides information about the literature survey conducted for this research work under various topics such as Service oriented architecture, web service, Legacy System and Legacy modernization approaches.

Chapter 3 outlines the methodological framework. It roughly presents the theory and practical details and illuminates the main workings of methodological framework.

Chapter 4 explains the Research Methodology used to implement the methodological framework discussed in chapter 3 for all the discussed modules.

Chapter 5 provides the implementation of the methodologies to design the proposed Service Oriented Architecture Framework.

Chapter 6 outlines the results and discussions of the proposed approach.

Chapter 7 provides conclusion of the proposed research work and future directions possible in this research area.