Design of a Communication Infrastructure for GUISET Services based on Multiple Enterprise Service Buses

Themba Shezi

(200702908)

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Science in Computer Science

Department of Computer Science, Faculty of Science and Agriculture

University of Zululand

KwaDlangezwa 3886

RSA

Supervisor: E. Jembere

Co-Supervisor: Dr. J Oladosu

2013

DECLARATION

I, Themba Shezi, declare that this dissertation represents my own work and that this work has not been previously submitted at any university or other institution of tertiary education. All sources of information used in this work have been acknowledged.

Themba Shezi

Signature_____

DEDICATION

To my late father, John Shezi

ACKNOWLEDGEMENT

First, I would like to thank the Almighty God, without his enduring mercy I could not have achieved anything.

Secondly, I would like to express my gratitude to supervisors Mr E. Jembere and Dr J Oladosu for their support and guidance throughout to make this research work a reality. I was very fortunate to have them as my supervisors. I would also like to extend my special thanks to my Prof. M.O Adigun and Prof. S.S Xulu for their fatherly support. I would also like to thank my fellow researchers and friends L. Nkosi, S.C Makhaye, S.K.S Ngwenya, M. T Nene,S. W Dlamini, S. Cebekhulu, P.T. Cwele and N.M Gumbi, thank you for all the support and motivation you gave me during the course of this work. Thanks to all members of the Centre for their support and assistance. In particular, P. Mudali, P.Tarwirei, B. Mutanga, A. Alaba, E. K Olatunji, M.V Shabalala, S. Fatyi, O Kayode, N. Sibeko, Z. Ndlela and N. Mdletshe.

I would also like to thank my mother and the whole family for their love, understanding and support.

Last, but not least, I would like to express my sincere appreciation to Miss P. Z Gumbi for her support throughout this study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List ofPublications

- **Shezi T**., Jembere E., Adigun M.O., Nene M.T. (2013). Enabling Dynamic Service Discovery and Composition in the Enterprise Service Bus. In Proceedings of SATNAC 2013 Conference, Stellenbosch, September 2013.
- **Shezi T**., Jembere E., Adigun M.O., Nene M.T. (2012). Analysis Of Open Source Enterprise Service Buses Toward Supporting Integration In Dynamic Service Oriented Environments. In Springer Link (LNICST series)for EAI Conference on e-Infrastructure and e-Services for Developing Countries
- **Shezi T**., Jembere E., Adigun M.O. (2012). Performance Evaluation of Enterprise Service Buses Towards Support of Service Orchestration. Proceedings, PSR Centre: International Conference on Computer Engineering and Network Security (ICCENS), Dec 26-27, 2012, Dubai, pp. 260- 264.
- **Shezi T**., Jembere E., Adigun M.O (2011). Towards Developing Failure Tolerant Communication Framework for GUISET Services. In Proceedings of SATNAC Conference, East London, Sept 2011.

ABSTRACT

In recent years, Service Oriented Architecture (SOA) has become a paradigm for enabling more efficient and flexible business processes in a service-based economy. The significance of this paradigm results in many organizations moving their businesses and making them available as online services so that they can be accessed ubiquitously by anyone connected to the network. The idea is to increase level of resource sharing and collaboration among geographically dispersed individuals/organizations. One of the successful SOA implementation that has recently received a lot of attention is the Enterprise Service Bus (ESB). ESB provides a key infrastructure that support guaranteed event handling, durable messaging, and data transformation capabilities that are needed by SOA environments. The success of ESB resulted in many ESB products being implemented and offered as both commercial and open source integration solutions. However, these products offer different approaches towards achieving ESB capabilities. Therefore, selecting the most suitable ESB becomes a challenging task, not only because there are many factors to consider in this selection, but also owing to the relationships between these factors and requirements of a particular integration scenario.

There are many research efforts that have attempted to assist in ESB selection. They only consider evaluation of ESB products against given integration requirements. These evaluations are only useful when there is an ESB product that best support all the integration requirements of a given environment. This is hardly the case because ESBs perform well in some capabilities and worst in others. It is, therefore, believed that multiple ESBs can be integrated to get the best of individual ESBs that can give better performance compared to a single ESB. On the backdrop of the foregoing, this work considered GUISET integration requirements and investigated the validity of the above mentioned belief by integrating multiple ESBs to work together as a

federation. Federation of ESBs allows each ESB to be used for the capability it best supports. Key capabilities investigated in this study are Service Discovery and Composition. An investigation was carried to find out which among the three (3) ESBs (ServiceMix, Mule and JBoss) considered best supports each of the afore-mentioned capabilities. The results showed that ServiceMix has the best support for Service Composition while JBoss has the best support for Service Discovery. These findings were then used for empirical evaluation of Directly Connected, Hub-Spoke and Brokered ESB Federation patterns, with each ESB providing the capability it best supports to the federation. Directly Connected ESB federation pattern outperformed the other patterns. We then compared the performance of Directly Connected ESB Federation and ServiceMix ESB to determine whether ESB federation has better performance compared to a single ESB. The results showed that ESB federation has better performance in terms of response time and throughput compared to a single ESB.

CHAPTER ONE
INTRODUCTION

1.1 Introductory Background

In today's world, businesses either small or large are striving for growth and competitiveness in order to be on the cutting edge. As a results business requirements and processes change frequently. To retain its agility and adaptability, businesses must ensure that their supporting IT systems and resources respond quickly to the changing business needs. The most basic needs of any business include cutting costs, quicker response to customer needs, integration of different systems across, the organization collaboration with other businesses through Business to Business (B2B) and Business to Consumer (B2C) interactions and achieving greater Return on Investment (ROI) (Fakorede,2007).

In response to the dynamic business needs, companies are required to build technologies from scratch, integrate applications across incompatible platforms and manually coordinate business process execution. None of these solutions are efficient because they are prone to error, very costly, difficult to maintain and could result to duplication of effort. While this is the case, business climate requires business processes to be automated to enable easy composition and integration of different systems. In addition, systems must be developed in such a way that they are supported across heterogeneous environment such as various operating systems, hardware, programing languages and middleware. One solution that meets these requirements is Service Oriented Architecture (SOA) (Papazoglu and Heuvel, 2007).

SOA has become a paradigm that allows interoperability between heterogeneous systems. SOA is acknowledged as architecture for integrating more complex systems at reduced cost and developing loosely coupled applications by means of services. Services are self-contained, well

defined and independent software component that defines a certain business function. Web service standards are the key enabling technologies for SOA. These standards include Web Service Definition Language (WSDL) which is an XML-based language for defining web services, Universal Description, Discovery and Integration (UDDI), which act as a repository for storing information about web services and SOAP which defines message structure for communication exchange over the Internet.

The significance of SOA paradigm in enabling efficient and effective business processes in the service-based economy results in many organizations moving their businesses to making them available as online services so that they can be accessed ubiquitously by any entity connected to the network (Serhani, 2010). The idea is to increase level of resource sharing and collaboration among geographically dispersed individuals and organizations. However this shift has partially reached Small, Medium and Micro Enterprises (SMMEs) especially in African countries due to the cost associated with the underlying infrastructure that realize this new way of doing business (Adigun *et al.,* 2006; Bauler *et al.,* 2006; Zdravkovic *et al.,* 2007). Therefore, to address this issue and others, an open dynamic service oriented and grid environment called Grid-based Utility Infrastructure for SMME Enabling Technologies (GUISET) was proposed in (Adigun *et al.,* 2006).

GUISET aims to leverage on the existing computing paradigm such as Service Oriented, Grid and Utility computing to provide affordable technologies to SMMEs. The idea is to provide a platform that would allow SMMEs to share information and resources thereby helping them to sell their products online and increase their trading territory without spending much on the technology. Chapter Two presents more details about GUISET.

The SOA implementation based on endpoints alone falls short of the key enabling infrastructure that support data transformation, guaranteed event handling, durable messaging and orchestration of multiple services using workflows (Genender, 2006). The importance of these additional requirements led to the concept of Enterprise Service Bus (ESB).Figure 1.1 shows the path from business needs to the ESB product. As previously mentioned, SOA is an architecture that delivers flexible, interoperable and cost saving approach towards achieving business needs. While, on the other hand, Web service is the key enabling technology for SOA architecture. ESB is an open, standard-based integration infrastructure designed to enable the implementation, deployment and management of SOA-based solutions with a focus on assembling, deploying and managing distributed SOA (Papazoglu and Heuvel, 2007).

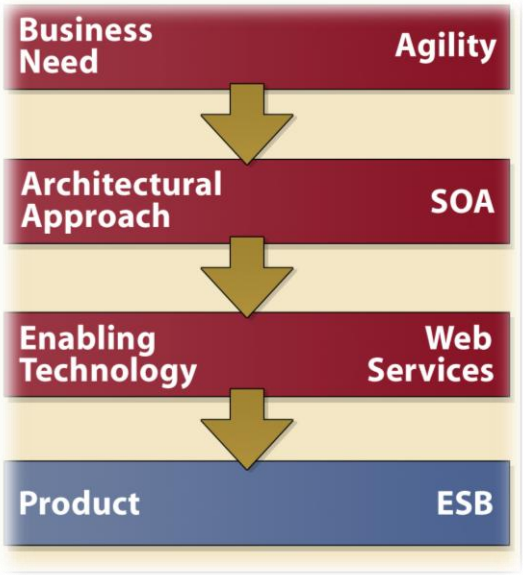

Figure 1.1: The path from business agility to the ESB (Cape Clear, 2005)

ESB integrate heterogeneous applications by providing a standard-based integration platform that combines web service, messaging routing, data transformation and service virtualization. Message routing is the core capability of ESB and it is based upon a number of factors, such as message content, message header and transport type. Data transformation is another capability of

ESB that enables data to be changed from source format to the one required by the destination application. Data transformation support loose coupling of communicating applications. Thus an ESB is responsible for transporting data, transforming it and routing it to the appropriate endpoint service (Goldshlager and Zhang, 2005; Ziyaeva *et al.,* 2008).The industrial success of ESB technology resulted in many products being implemented and offered as both commercial and open source products. Commercial products include Biztalk server, BEA Aquatic Service Bus, IBM WebSphere, Tibco, and Oracle ESB. Open source market includes Mule, ServiceMx, WSO2, JBoss, Petals and Open ESB (Vollmer *et al.,* 2011). These product implementations provide different approaches towards realizing ESB capabilities so the problem of how to select the most suitable ESB product for a given business requirements is critical (Kruessmann *et al.,* 2009). Not only because there are many factors to consider in the selection, but also owing to the relationships between these factors and the requirements of a particular integration scenario. This work argues that a centralized single ESB cannot best support all the GUISET integration needs owing to the fact that ESBs use different approaches towards achieving a certain integration capability. In view of the foregoing, this work aimed to do the following:

i). Evaluate the existing ESB implementations towards supporting GUISET integration requirements.

ii). Based on the results obtained from the evaluation, the work further determines an infrastructure that would best support GUISET integration requirements.

1.2 Statement of the Problem

Currently, there exist many ESB implementations on the market, so selecting one ESB to support all the integration requirements is a challenging task. The existing literature suggests qualitative and empirical evaluation of ESBs against the given integration requirements. This evaluation can

only be sufficient if there is a single ESB product that best support all the required integration capabilities. However, it is possible to have multiple ESBs offering the best support for different integration capabilities required. For example, ESB1 might perform well when considering data transformation and intelligent routing but comparably worse on service composition. Therefore, this work will explore the possibility of designing an infrastructure that best support all the integration requirements needed by GUISET even though the above mentioned problem exists.

Recent research work has demonstrated the limitations of employing single centralized ESB to support integration in large scale environments (Callaway *et al.,* 2008; Nair S., 2009; Baude *et al.,* 2010). These limitations stem from the fact that today's enterprises are distributed in nature, so ensuring centralized single ESB in a geographically dispersed environment is challenging. In addition, a single ESB might impose scalability issues which are particularly important for any pervasive integration. Regardless of the nature of the business; these limitations are driving today's systems integration towards federated ESB models. Federated ESBs consist of multiple ESB domains working together to form a single, logical ESB. There exist three federation patterns that have been proposed, namely: Directly Connected, Hub-Spoke and Brokered ESB Federation. However, none of the existing work known has investigated the performance of these ESB federation patterns.

Therefore, in light of addressing the above mentioned challenges in the GUISET context, this research work investigates the following:

How can the existing ESBs be integrated to best serve GUISET integration requirements?

(i) Which integration capabilities will GUISET services need?

(ii) Which existing ESB solution can best support each GUISET integration need?

(iii)How can the existing ESBs be integrated to ensure that each ESB is used for a capability it best supports in a GUISET environment?

1.3 Research Goal

The goal of this research project was to determine the best strategy that can be used todesign a communication infrastructure that best support the GUISET integration requirements.

1.4 Research Objectives

(i) To investigate which integration capabilities that GUISET services will need through creation of GUISET usage scenarios.

(ii) To conduct literature survey on existing ESB solutions.

(iii) To determine which ESBs can best support a given GUISET integration need through empirical comparative analysis of the ESBs.

(iv) To determine the best strategy of integrating multiple ESBs for GUISET environment through empirical comparative analysis of the existing ESB federation patterns.

1.5Research methodology

The work to be conducted in this study is largely experimental with a lot of activities that are dependent on each other. To give a better understanding on how this research was conducted Section 1.5.1 presents the research design developed for this study. Section 1.5.2 discusses the specific research methods used in this study and how they were applied to get scientifically sound results from this study.

1.5.1 Research Design

This section summarizes the overall study design, activities and the extent of data resulting from this research. Research design presented in this section was used to provide a framework for

designing a systematic study that would address the study's research goal, objectives and questions. In a sense, the research design is a blueprint of the research that deals with at least four problem; what questions to study, what data are relevant, what data to collect and how to analyze the results (Yin, 2008).

This work argues that that the conflicting ESB selection criteria can be tackled by integrating multiple ESBs with each ESB responsible for the capability it best support. The purpose of this work was to determine the best strategy that can be used to integrate multiple ESBs such that GUISET integration requirements are best supported. Towards this aim, it was then important to investigate which integration capabilities are needed by GUISET environment. This investigation was based on the GUISET usage scenario. Scenario is defined as the combination of actions or events that allows an organization to analyze complex business processes. Scenarios are widely used by organization of all types to understand different ways that future events might unfold (Gregory and Duran, 2001). The GUISET integration requirements obtained from the scenario was then used to investigate which ESB has the best support for each GUISET integration requirement. This investigation was used to ensure that when integrating multiple ESBs, each ESB is used for the capability it best supports. Qualitative and empirical analysis of ESBs was used for the above mentioned investigation.

- Qualitative analysis involved collecting evidences from vendors' documentations and publicly available evaluations of the ESBs published by independent entities. The data collected was focused on the efforts made by ESBs towards supporting each of the GUISET integration requirements. Having only data was not enough to make conclusions on ESB capabilities, therefore, a Multi-Criteria Decision Analysis method known as Analytical Hierarchical Process was used to further analyze data. Using AHP,

the GUISET integration requirements were ranked according to their importance towards supporting integration in a GUISET and other service-based environment. The use of multiple sources of evidence has been suggested as one of the several methods that can be used to improve the quality of data and research findings (Patton, 2001; Yin, 2008). To verify the findings obtained using AHP, this work employed an empirical analysis method.

- The empirical method was used to comparatively evaluate the performance of ESBs given the GUISET integration capability. The ESBs were deployed in a simulation environment and the implementation allowed us to observe scalability, response time and throughput as performance metrics. Although the results obtained from these metrics provides the valuable information on how each ESB performed, the data was further analyzed using statistical method known as Analysis of Variance (ANOVA). ANOVA allowed us to observe the significant difference in the performance of ESBs.

The use of the above research methods provide the answer to the second research question of this study as it seeks to determine the best ESB for each GUISET integration requirement. The third and last research question seeks to determine the best strategy for integrating multiple ESBs. As an attempt to answer this question, the ESB federation approach was used. ESB federation allows multiple ESBs to work together to achieve integration objectives. There are three ESB federation patterns that have been proposed and these are Directly Connected, Hub and Spoke and Brokered ESBs. This research presented the comparative empirical evaluation of these patterns with the aim of investigating their performance tradeoffs. The metrics used for performance evaluation were scalability, response time and throughput. ANOVA was used to analyze the data collected during the experimentation. The ESB federation pattern that had the

best performance was then compared with the single ESB that was found to be closer to meeting the GUISET integration requirements. This comparison was to find out whether ESB federation has better performance compared to a single ESB.

1.5.2 Research Methods and Techniques

From the foregoing discussion, it can be clearly seen that the main research activities in this study were comparing ESBs' performance on each GUISET integration requirement, comparing different ESB federation patterns, and finally benchmarking the best performing ESB federation pattern against a single best performing ESB. The discussion that follows presents how standard research methods techniques were used throughout these activities.

1.5.2.1Literature Survey

Intensive literature survey was conducted with the aim of finding the background concepts of enterprise service bus, multiple ESB integration patterns and GUISET. Survey on ESBs was channeled to focus on the capabilities, characteristics, market visibility and configuration details with the aim of finding the most popular and comprehensive ESBs. The knowledge gained from understanding of GUISET and its background was used to formulate usage scenario so as to ascertain GUISET integration requirements.

1.5.2.2 Qualitative Comparative analysis of ESBs

This method involved doing literature survey on each ESB towards supporting each GUISET integration requirements. Having literature alone was not enough, so Multi Criteria Decision analysis (MCDA) technique was used to analyze the results obtained from literature survey. In addition, MCDA was used to determining which ESB had the better support for each integration

requirement needed by GUISET. Moreover, GUISET integration requirements were ranked according to their importance.

1.5.2.3Experimentation: Empirical Comparative Analysis

This method enabled us to conduct the experiments involving the actual deployment and configuration of ESBs services. Two sets of experiments were conducted. These were: (i) experiments to evaluate the performance of each ESB on Service Discovery and Service Composition, and (ii) Comparison of ESB federation patterns in the context of GUISET integration requirements.

a. Empirical Comparative Analysis on ESBs

In this analysis, ESB performances were compared with the aim of finding ESBs that best support Dynamic Service Discovery and Service Composition. Performance metrics which included service response time and throughput were used for evaluation. Scalability was also tested to check how ESB respond to load increase. The result was used to select ESBs that perform best in a given GUISET integration scenario.

b. Empirical Comparative Analysis on ESB Federation patterns

This analysis used the results obtained from (a) above. Each ESB was used for the capability it was found to best support (a) in each of the identified ESB federation patterns. The results obtained from these experiments were then used to recommend the best federation pattern to integrate two or more ESBs for a GUISET-like environment.

1.6 Dissertation Synopsis

The rest of the dissertation is organized as follows:

Chapter Two presents the background concept of this research work. These concepts include GUISET and its integration requirements, enterprise integration, Enterprise Service Bus and Service Oriented Architecture. In addition to these background concepts, we also analyzed and selected the ESB implementations that would be used throughout this work.

Chapter three presents the literature review on ESB evaluations. This review was restricted to only evaluations related to GUISET integration requirements. The ESB analysis in the context of their support for GUISET integration requirements is also described. Chapter Four presents the empirical evaluation of ESBs against the GUISET integration requirements. These evaluations were done to find the best ESB that support each integration requirement.

Chapter Five then presents the concept of ESB Federation and also gives the overview of ESB federation patterns. The implementation of each pattern was presented with the aim of investigating their performances. Finally, the performance results and analysis of ESB federation patterns were presented.

Chapter Six concludes the dissertation. The recommendation for future work is also presented.

CHAPTER TWO

BACKGROUND OF ENTERPRISE SERVICE BUS CONCEPT

2.1 Introduction

Over the past years, businesses were required to cope with the rapidly changing markets which included new competitive pressure and trends that require information systems to respond quickly towards supporting new business models and requirements (Ortiz, 2007). With changes becoming more and more frequent it was not unusual for an enterprise to have developed and collected large number of applications over time. These applications were usually diverse in nature. In addition it was also a rare case if not none that a single application can cover the needs of the whole organization. Therefore, this was one of the justifications for having small number of applications developed over time. To respond quickly to the rapidly changing market environments, enterprise integration emerged to address the enterprise needs of integrating business applications into a single system to allow data to be shared internally, with the third party and customers.

As indicated in Chapter One, the aim of this research was to determine the best strategy that can be used to design a communication infrastructure that best support GUISET integration requirements. This Chapter presents the background of key concepts in this study. GUISET and its integration requirements are presented in Section 2.2.This is followed by the discussion of approaches that have been proposed towards enabling enterprise integration (Section 2.3). Among these approaches, Enterprise Service Bus has been the most successful and widely used due to its use of open standards and its capability of integrating infrastructure that allows fundamental Web service and SOA concepts to coalesce. Section 2.4 presents SOA concepts and the web services technology. To qualify a product to be called an ESB, it should feature certain

core capabilities, these capabilities are presented in Section 2.5. In Section 2.6, we present the summarized comparison of the enterprise integration approaches. The popularity of ESB led to many products being implemented and made available on the market. These products are architecturally different and they use different techniques towards achieving integration capabilities. In Section 2.8, we evaluate five ESB products against general criteria with the aim of selecting the three most comprehensive ESB products that will be used for evaluation against GUISET integration requirements. Section 2.8 presents the details of the three selected ESB products. Finally, Section 2.9 presents the chapter summary.

2.2 The GUISET Project

GUISET stands for Grid-based Utility Infrastructure for SMME Enabling Technologies. GUISET is mainly motivated by the significance of SMMEs in stimulating economic growth and generating employment that would results in local development especially in African countries. SMMEs are challenged by unaffordable ICT infrastructures that would allow them to easily expand their trading territory. This is believed to be one of the impediments to growth of SMME. To address this issue GUISET was proposed (Adigun *et al.*, 2006). GUISET aims to provide affordable technology to SMME through utility approach to service delivery. The idea is to provide a platform that would allow SMME to share information and products thereby helping them to market and sell products online without spending much on technology. The question that arises is how would this infrastructure ensure affordability? Using GUISET infrastructure, SMMEs would register for only services they want and only pay as they use those services. To make this possible, GUISET leverages on the existing computing paradigms including Grid, Service Oriented Computing and Utility Computing.

- Grid computing allows GUISET services/resources to be distributed and shared across multiple geographic and administrative domains
- Service Oriented Computing would allow GUISET components to be offered as services that can be easily assembled to form new applications thereby increasing reuse of services.
- Employing utility computing would ensure that all GUISET services are used and SMME pay as they use these services. This model has been acknowledged as an efficient way of offering service. Cloud computing is an example that utilized this model.

2.2.1 The GUISET Architecture

Figure 2.1 depicts the GUISET architecture. The architecture is divided into three layers. These layers are Multi-Modal interfaces, Middleware layer and Grid Infrastructure Layer. The architecture also maps the relationship among different entities of GUISET.

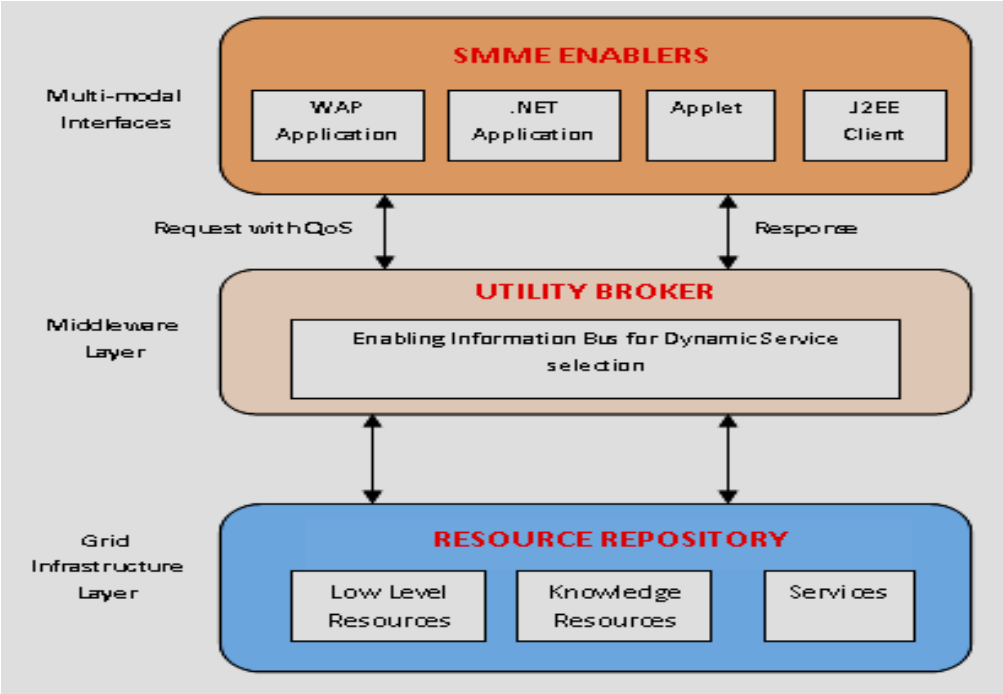

Figure 2.1 GUISET Architecture (Adigun *et al.*, 2006)

Multi-modal interface layer contains services that are exposed as interfaces in this layer to allow any client regardless of the underlying technology to have access. This layer establishes the necessary interactions with the external entities (including SMMEs) and middleware layer.

Middleware layer contains information bus which enables the dynamic service discovery and selection. Therefore, this layer describes the glue between Multi-modal interfaces and Grid infrastructure layer.

Grid infrastructure layer is where GUISET services and resources reside. In this sense services implement the business logic that is exposed as interface in the Multi-modal interface layer.

This study focuses more on Middleware layer where integration capabilities should be supported due to diversity in terms of technology in such an environment. As previously mentioned that this layer provides glue between the other layers, therefore, there is a great need to ensure that the infrastructure employed at this layer is most suitable for necessary interaction to take place effectively. For example, in the next section a typical GUISET scenario is given to show interactions between different services.

2.2.2 GUISET Integration Requirements

To ascertain GUISET integration requirements, we consider a Stock purchasing application offered by GUISET to its customers. To successfully complete the purchasing process a number of services need to communicate and exchange messages. These services include order validation which is responsible for validating whether order has all the necessary information for it to be processed. On the other hand, there is stock control service which checks and reserve items purchased. Finally, the shipping service is required to deliver the purchased order to the

relevant destination. From this GUISET usage scenario the following integration requirements were gathered:

- An integration middleware is required through which all GUISET components and services can have reliable message exchange, therefore minimizing downtime of services
- A facility for mapping data format from source application to the one required by destination application to avoid incompatibilities between interacting applications.
- Intelligent routing of messages between services. GUISET would offer Quality of Service (QoS) aware services. Services with similar functionalities would be selected based on their QoS.
- Support for service composition to allow service reusability by forming new application from existing services. For example Stock purchasing application is formed by existing services which can be used for other functions as well.
- Support for dynamic service discovery. For example whenever the requested service is unavailable it should be possible to substitute this service without affecting the client application.

As previously mentioned, the main aim of this work was to evaluate the existing integration mechanism in support of GUISET integration requirements. Section 2.3 overview different enterprise integration approaches.

2.3 Enterprise Integration

Enterprise Integration is a technical field which focuses on issues such as system interconnectors, electronic data exchange, and product data exchange in distributed computing environments (Lima, 2011). Enterprise Integration aims to connect and combine people, processes, systems

and technologies to ensure that the right people and the right processes have the right information and the right resources at the right time (Brosey *et al.,* 2002). A typical scenario where application integration becomes major concern is when the purchase order application which is generated using the company's e-commerce application needs to query a customer loyalty database to determine whether the customer has done enough business with the company in the past to merit for special pricing or treatment. To successfully accomplish this process more than one applications must effectively and flexible communicate and exchange data.

To address issues of enterprise integration, three approaches has been proposed. These approaches are Enterprise Resource Planning, Enterprise Application Integration and Enterprise Service Bus. ESB approach is based on Service Oriented Architecture (SOA), so this concept is also introduced for the better understanding of this approach.

2.3.1 Enterprise Resource Planning (ERP)

Back in the days ERP was the dominating enterprise integration approach due to its tightly integrated solution for organization's information system needs. ERP is a business management system that comprises of integrated sets of comprehensive software modules that assist to successfully implement, manage and integrate all business functions within an organization (Themistocleous *et al.,* 2004). With ERP, important parts of business are managed. These parts include product planning, parts purchasing, maintain inventories, interacting with suppliers, providing customer service and tracking orders. ERP has the ability to centrally facilitate flow of information between all supply chain processes in an organization. However, the architecture of ERP follows the centralized management model. Moreover, applications communicate directly with each other in a point-to-point method, which can lead to maintenance cost increases and difficulty to reuse application. In addition, implementation of ERP solution is extensive, lengthy,

and expensive. The startup costs for large organizations were typically measured in millions of dollars (Shehab *et al.,* 2004). Hence, with these drawbacks of ERP, the research for alternative or supplementary technology was done and EAI was proposed.

2.3.2 Enterprise Application Integration (EAI)

EAI was proposed not only to improve the drawbacks of ERP but also to enable effective integration of both intra and inter-organization system by allowing applications to communicate with the broker using adapters. According to Soomro and Awan (2012), EAI can be defined as the "combination of processes, software standards and hardware that results in the seamless integration of two or more applications". EAI is unlike the ERP which consume more time and money by requiring the rewriting of code for integration, EAI uses a special middleware that act as the broker between applications to enable seamless system integration. Typically these middleware are Message Oriented Middleware (MOM) products. MOM allows communication between applications to be asynchronous, but also can be synchronous. Asynchronous communication offers an advantage by allowing a sending application to send message regardless of whether the receiver is up and running. The message is stored by the broker and can be received some other time. This communication is efficient because the sender is not blocked during the communication and it can be processing other things while waiting for response.

There are two basic architectures for EAI

- **Hub and Spoke** – This integration technique has a centralized broker (Hub) and adapters (Spoke) where the transformation and routing of messages occurs. Spokes connects applications to the Hub. Moreover, Spokes transform data to the format that Hub understands and vice versa. Hub brokers all messages and determines the appropriate

destination. Hub and Spoke offers an easy to manage integration, however scalability is the major concern. In addition to scalability, broker is the performance bottleneck and the single point of failure.

- **Bus** – The drawbacks of Hub and Spoke introduced serious limitations and projects that adopted it were not successful (Soomro and Awan, 2012). This led to the new integration solution based on the Bus. A Bus is the centralized messaging backbone where applications publish and consume messages using adapters. The main difference with the Hub and Spokes is that the Bus architecture enables integration engine that performs transformation and routing to be distributed in the application adapters. This architecture achieves a better scalability than Hub and Spokes but it difficult to manage.

The major problem with EAI solutions is that they often use proprietary protocols which lack standards to improve interoperability with other solutions. Lack of interoperability can lead to isolated EAI- based infrastructures. Proprietary solutions come with lot of functionalities packaged as the product. Most of these functionalities are never used but their costs are included during purchasing. So organizations are constantly incurring unnecessary additional costs.

2.3.3 Enterprise Service Bus (ESB)

The early adoption of EAI solutions specifically Bus model led to ESB. In short ESB is a collection of middleware services which provides integration capabilities. The difference between ESB and EAI is that ESB is based on open standards which eliminate technology lock-in. ESB is designed to be the main implementation of Service Oriented Architecture, so it has all the advantages that SOA offers (Menge, 2007). ESB is the state-of-art in enterprise integration and SOA. Hence before looking deep into the ESB paradigm, it is necessary to give a brief discussion on SOA.

2.4 Service Oriented Architecture (SOA)

SOA was introduced to enhance the domain of information system development by addressing requirements of loosely coupled, standard-based and protocol independent communication between distributed systems/applications (Papazoglou and Heuvei, 2007). SOA is defined as a set of guidelines, architectural patterns, acceptable and recommended practice for defining business functions as independent services described using Web Service Description Language (WSDL), published on the Universal Description, Discovery and Integration (UDDI) and communicated via SOAP messages over the network as shown represented by publish-find-bind shown in Figure 2.2 below. At the core of SOA, are services which are self-describing, open components that support rapid, low-cost composition of distributed application.

The adoption of SOA allow developers effectively overcome distributed enterprise computing challenges that include application integration and transaction management while enabling multiple heterogeneous platform and protocols to have access to devices and legacy systems. Therefore SOA eliminate barriers so that application integration can run seamlessly.

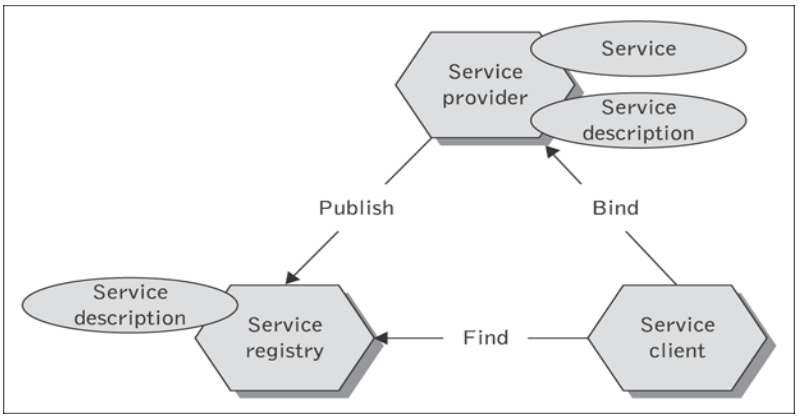

Figure 2.2: SOA publish-find-bind architecture

Web service technology is the key enabler for SOA, for the better understanding of the concept an overview of this technology is given below.

2.4.1 The Web Services

Web service is defined by W3C as a service that supports web standards. More specifically, it is exposed through network and can be accessed by any other application connected to the network. The accessibility of web service is completely platform independent because of its formal description that is defined using WSDL. The communication between web services and other applications is made possible by SOAP protocol transported via HTTP. Web service is an abstract notion which should be implemented by either concrete software or an interface to it. The flexibility and open standard compliant that web services offer has made it to be widely supported by major software vendors (Papazoglou, 2008). Key web service standards are as shown in Figure 2.3. This work focuses on SOAP, WSDL and BPEL standards of web services. In addition, this work presents web service discovery and selection as the concepts that allow web services to be discovered and selected by other services, applications and business processes.

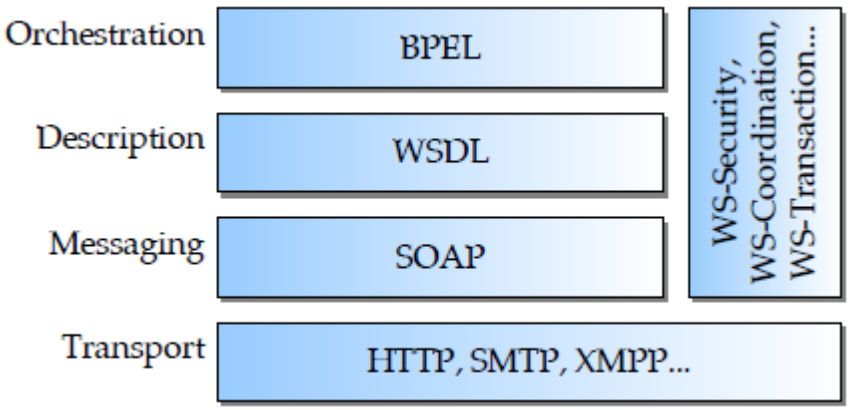

Figure 2.3: Web service standards (Papazoglou, 2008)

2.4.2 SOAP

SOAP is the XML based communication protocol which allows the exchange of messages between distributed applications. SOAP defines a message structure that can be transported by a

number of protocols including HTTP, Java Message Service (JMS), XMPP and SMTP. However HTTP is the most used protocol for transporting SOAP messages.

2.4.3 Web Services Description Language (WSDL)

WSDL is an XML language used to describe the publicly available functions and methods of a web service. WSDL interface is divided into two parts namely abstract and concrete part. Abstract defines the general information about the service while concrete gives technical information to assist in finding and using the web services.

2.4.4 Service Discovery

Web service technology has gained a lot of attention in developing distributed applications and systems on the Internet. The rapid growth of published web services makes their discovery process to be a complex task. The most important part of any SOA environment is how other applications find information about a service. This is typically provided by service discovery. There are several steps involved in discovering a web service.

- The requester describe the service it requires and then send it to the discovery service to search and return the information about service (s) that match the requester's description. The discovery service either return full service description or a list of matching services
- Upon receiving matching service, the requester and the provider agree on the way they are about to communicate.
- Then the two entities might start exchanging SOAP messages according to the functionalities required by the requester.

One of the most successfully technology for service discovery is UDDI service registry.

2.4.5 UDDI Service registry

UDDI is a repository for storing information about the web services (Clement *et al.*, 2004). Service providers publish the information about their services in the UDDI. These services are discovered and used by any interested parties. The information stored on the UDDI includes business and service description. Business description consists of service provider's name and other relevant information. Service description consists of web service name, functionalities offered and technical information on how to access the web service. Figure 2.4 illustrates the data structures that exist in the UDDI registry and they are businessEntity, businessService, bindingTemplate and tModel.

- *businessEntity* describe information about the service provider that has published the Web service
- *businessService* describes the functionality provided by Web service
- *bindingTemplate* provide description of the details on how to access the web service
- *tModel*describes the technical information that include categorizations and specifications of a web service.

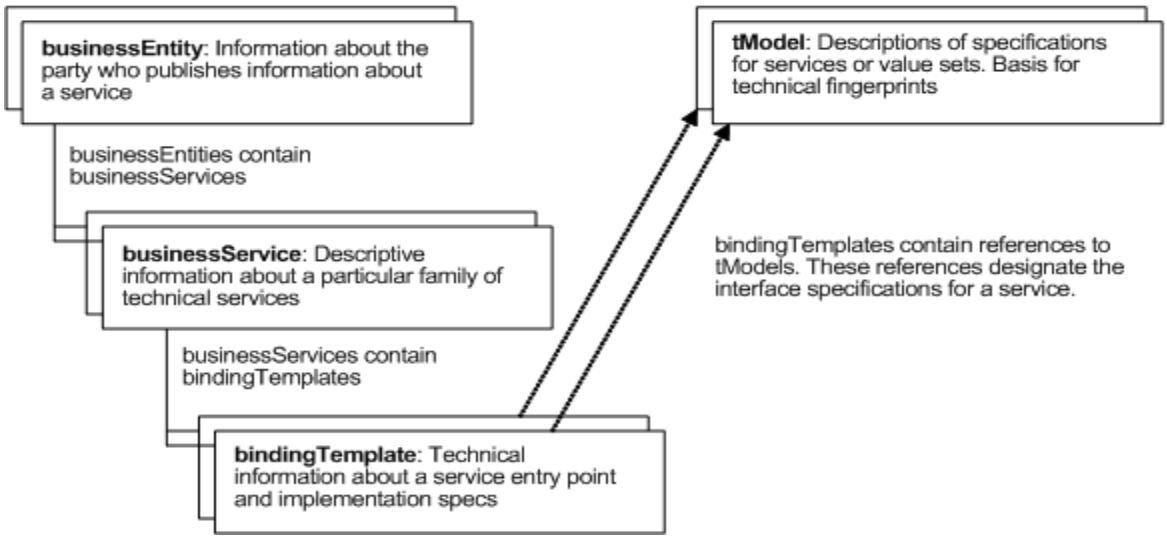

Figure 2.4: UDDI core data structure (Clement *et al.*, 2004)

2.4.6 Service Selection

Service selection is the process of selecting the appropriate service based on the specified requirements. The wide acceptance of SOA can result in many services that provide the same functionalities being published in the UDDI registry. So the major problem is how to select a single service from the list of matching services. Quality of Service (QoS) has been proposed as the differentiating factor among services that provides the same functionalities. QoS is the non-functional attributes of the web service. QoS attributes include web service performance, dependability, price and reputation (Mareno and Raffeala, 2010). Unfortunately the current implementation of UDDI service registries does not store QoS information. Blum and Fred (2004) suggested that QoS information of the Web service can be stored inside UDDI registry by using categorization structure of the tModel. This approach has been widely used by a number of authors. Figure 2.5 shows tModel structure storing the QoS information as proposed by Blum and Fred.

```
<tModel tModelKey="mycompany.com:StockQuoteService:
PrimaryBinding:QoSInformation"" >
  <name>QoS Information for Stock Quote Service</name>
  <overviewDoc>
    <overviewURL>
      http://<URL describing schema of QoS attributes>
    <overviewURL>
  <overviewDoc>
  <categoryBag>
    <keyedReference
              tModelKey="uddi:uddi.org:QoS:ResponseTime"
              keyName="Average ResponseTime"
              keyValue="fast" />
    <keyedReference
              tModelKey="uddi:uddi.org:QoS:Throughput"
              keyName="Average Throughput"
              keyValue=">10Mbps" />
    <keyedReference
              tModelKey="uddi:uddi.org:QoS:Reliability"
              keyName="Average Reliability"
              keyValue="99.9%" />
  </categoryBag>
</tModel>
```

Figure 2.5: QoS information stored on UDDI tModel data structure (Blum and Fred, 2004)

The categoryBag has keyedRefence field where QoS information about the web service is stored. The structure has tModelKey which uniquely differentiate QoS attributes of the service, keyName stores the name of QoS attributes and keyValue is the actual value of the QoS.

2.4.7 Web Service Composition

In SOA environments, to enable business-to-business and enterprise level application integration, composition of web services plays an important role. Sometimes it is always possible that a single service cannot be able to fulfill the requester's need, therefore different web services are combined through composition method to achieve a specific business goal. This is the core of SOA as it aims to promote service reuse by creating new services from the existing ones. There are two methods towards service composition, namely service choreography and orchestration.

2.4.7.1 Web Service Choreography

Web service choreography is a composition technique that is based on collaboration. In this technique all web service participating in the composition are treated as equal entity with no central control of the execution. Participants need to know exactly when to become active and which service to collaborate with. This means all services need to have information about other services in the composition.

2.4.7.2 Web Service Orchestration

Web service orchestration approach is based on the central control that coordinates process execution. In this approach services participating on the process have no information about the other and they are not aware of their participation in the process. The most successful language for defining these processes is called Web Service Business Process Execution Language (WS-

BPEL). WS-BPEL is the XML-based language standardized for defining business processes. In the context of ESBs web service orchestration has been the most supported approach compared to web service choreography and there has been advancement of this approach including the BPEL engines that can be used to execute business processes defined using WS-BPEL. In view of the fore-going, the approach used for service composition in this work is focused on service orchestration.

2.4.7.2.1 Business Process Execution Language (BPEL)

BPEL is a process-oriented web service composition language. More concretely it defines the order in which web services need to be invoked. BPEL is not a fully featured programming language like Java simply becaused it was develpoed only to defines business processes. BPEL defined processes are executed by BPEL engines. The current version of BPEL is called WS-BPEL 2.0. This version was finally approved by OASIS in April 2007 (Arkin *et al.*, 2007) and it is hardly backward compartable with version 1.1 (Linthicum, 2006).

2.4.7.2.2 BPEL process Components

BPEL language define several components that have designated functionalities as shown in Figure 2.6and illustrated below

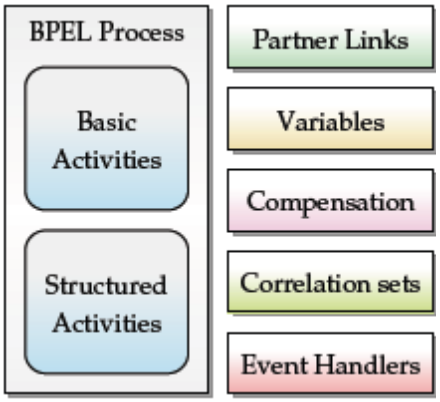

Figure 2.6: BPEL Components

Activities

BPEL activities are like operators of the language. They define steps involved for the process to complete its intended operation. There are two types of activities, basic and structured.

Basic Activities: These type of activities cannot contain any other activities. They represents an individual step in the process evolution. Some examples of basic activities include:

- <recieve> Waits for a message to arrive.
- <invoke> Invokes a one-way or request-response operation for a given partner.
- <assign> Update the value of a variable with new data.
- <reply> Send process response to the requesting application.

Structured Activities: The activities offer a way to structure the BPEL processes. They assist in mananging the flow of a process by enabling concurrent activities, fault handling and coordination whenever it is required. Some structured activities are:

- <sequence> Contains collection of activities to be performed sequentally one after the other.
- <flow> represent a set of activities that should run concurrently.
- <if> Specify the condition to select the execution of an activity.
- <pick> Waits for the occurrence of exactly one event for a set, then executes the activity associated with that event.

There are more activities for defining BPEL proceses and they can be found in (Arkin *et al.*, 2007)

2.4.7.2.3A simple BPEL process

Here we give a simple BPEL process called Hello_Process as illustrated in Figure 2.7 below. At first the process waits (<receive>) for an input message carrying the string "name" of a person. It then appends "Hello" before that name (<assign>), and finally send back the greeting message (<reply>). All of this code is a part of a structured activity <sequence> that ensures every activity execute in the correct order.

Figure 2.7: Sample BPEL process

2.5 Enterprise Service Bus and Service Oriented Architecture

Since the wide adoption of SOA as the paradigm that enables flexibility and agility required by business users and processes, organizations are paving way for this paradigm by refactoring their systems and data models. Executives have found SOA to be the solution that would allow business systems to react quickly to the ever changing business environment (Genender, 2006). Today, enterprises impose high demands on the integration solution due to high diversity of applications in terms of technologies, frameworks, protocols and design concepts.

ESB is an ideal solution for such demands as it is based on open standards and SOA. In this sense ESB is a standard based integration platform that combines web service technology, messaging, data transformation and intelligent routing together to provide flexible integration solution.

In order to support SOA, the ESB has to offer certain key capabilities. The following are the core of these capabilities (Davis, 2009):

- Message routing capability – The most important capability of an ESB is being able to route requests to their destination. With this capability the ESB helps to decouple interacting entities by relieving service requester from knowing physical location of service provider thereby achieving location transparency. ESBs use different mediation logic to determine the destination. These logics can be implemented statically (hardwired) or dynamically via content-based routing.
- Data connectivity and adapters – ESB should be able to support several open standards, such as SOAP, WSDL, UDDI, JCA (J2EE Connector Architecture), JBI (Java Business Integration) and different transport protocols like HTTP/s, JMS, TCP, SMTP/POP3, File batch, etc. In addition ESB should support conversion of these protocols to allow integration of applications that use different protocols. For example, requesting application might use HTTP to send request whereas destination application use JMS for communication. Therefore, achieving interoperability between multiple transport protocols.
- Communication styles –An ESB should support applications which use both synchronous and asynchronous messaging as communication style.
- Mediation capability – different ESB may use different mediation capabilities. However, in general mediation is rather used for situations where the flow is composed of simple reusable actions in a form of Enterprise Integration Pattern (EIP). This includes filtering, routing, transformation, etc.
- Process Orchestration –ESBs use different tools to support orchestration. This capability is rather used for complex workflows which involves invocation of external services and may

require human interaction. This is used for high-level service composition composed of short and long running business process. Web Service Business Process Execution Language (WS-BPEL) is the standard used to define business processes. To support execution of these processes most ESBs use different BPEL engines.

- Data Transformation – Typically two interacting services may understand different data formats, this is usually the case when legacy systems or external services are integrated. ESB should be able to transform data from one format to another to enable smooth communication between different applications. For example, an application which understands only XML messages may communicate with application which produces CSV data. XSLT is the most used transformer for XML based messages.
- Security – In the corporate sector security is very important, so an ESB should provide mechanism for authentication and authorization of incoming requests, encryption and signing their content.
- Service discovery – ESB should expose services so that they can be discovered either statically or dynamically by other services.
- Monitoring and Management – management and monitoring tools should be provided by ESB to enable central management of integration logic, service lifetime, etc.

2.6 Comparison of Enterprise Integration Approaches

Thus far, we have discussed three approaches towards achieving enterprise integration. Table 2.1 shows the summary of comparisons between these approaches. ESB and EAI are very similar, however, ESB is based on open standards and therefore, it is a more loosely coupled integration solution while EAI is based on proprietary standards that lead to technology lock-in.

ESB has been widely adopted as the cornerstone of today's information system integration due to the flexibility that this platform offers. So the rest of this work is based on ESB approach to enterprise integration. The next section presents enterprise integration solutions based on ESB approach.

Table 2.1: Comparisons of Enterprise Integration pattern

Evaluation Parameter	**ERP**	**EAI Hub and Spoke**	**EAI Bus**	**ESB**
Topology	Single/Multi Sever	Hub	Bus	Bus
Architecture	Centralized	Centralized	Decentralized	Decentralized
Integration Effort	High	Medium	Medium	Low
Coupling	Strong	Moderate	Moderate	Loose
Scalability	Low	Medium	High	High
Management	Easy	Easy	Complex	Complex
Development	Proprietary	Proprietary	Proprietary	Standard
Cost	High	Medium	Medium	Low

2.7 ESB Solutions

The industrial success of ESB technology led to many products being implemented and offered on the market. These products include commercial products like BizTalk server from Microsoft and IBM WebSphere from IBM and open source products like Mule from MuleSoft and ServiceMix from Apache. The recent adoption of open standards like JBI, SOAP, JMS, XML and many others led to open source ESBs receiving attention from the literature and business systems integration. Many open source ESB products are available in the market today, however this work considers only a few for evaluation based on their popularity in the literature. These

ESBs includes Mule, ServiceMix, JBoss, Open ESB and OW2 Petals ESB. In order to successfully design a communication infrastructure for GUISET services, there was then a need to first evaluate ESBs according to a given general criteria.

Defining General Criteria

The main aim was to evaluate the completeness of each ESB product. This investigation help determine which ESBs to include in the evaluation against GUISET integration requirements. Therefore we defined four general criteria as support for core ESB capabilities, quality of the documentation, active community support and market visibility.

Support for core ESB capability

In Section 2.5, we presented the core ESB capabilities, so it was important that an ESB of our choice should provide support for all these capabilities

Quality of the documentation

In this context, quality of the documentation refers to the availability of help resources such as user guide, books and tutorial that support a novice user to easily get started with the ESB product. Open source products have been previously criticized regarding their documentation. However, some vendors have made major efforts to ensure that their product has quality documentation.

Active Community

Open source ESB products are available for anyone to use. However during the process of using the product bugs are most likely to happen. Therefore, it is important that an ESB selected is supported by active community so that bugs can be quickly fixed.

Market visibility

Another important criterion is the market visibility which refers to the popularity of the ESB product. Popularity is measured against a number of available publications that mention the ESB product.

2.7.1 Assessing the ESBs

Considering support for core ESB capabilities, Open ESB still has a lot of improvement required especially when considering message routing, monitoring and management capabilities (kusak, 2010). The other four ESBs provide considerable support for core capabilities, however, in the area of service discovery there still a room for improvement for all ESBs. On the other hand, OW2 Petals has small number of community with 138 registered users, 5 administrators and 2 members (OW2 Middleware consortium, n.d). JBoss, Mule and ServiceMix were considered for the evaluation by Forrester Wave. Forrester Wave makes up the leading evaluation of both open source and commercial ESBs against 109 criteria. These three open source ESBs came up as strong performers against these criteria (Vollmer *et al.,* 2011). Therefore, in this work these ESBs were considered for evaluation against GUISET integration requirements.

Table 2-2: ESB evaluations against general criteria

Criteria	JBoss ESB	Mule ESB	Open ESB	OW2 Petals ESB	ServiceMix ESB
ESB Core capability	Good	Excellent	Weak	Good	Excellent
Quality of the documentation	Good	Excellent	Weak	Good	Good
Active Community	Excellent	Good	Good	Weak	Excellent
Market visibility	Good	Excellent	Good	Fair	Excellent

2.8 Overview of the selected ESBs

Availability of quite a number of ESBs on the market makes selection of one product to use a difficult task. However, due to the analysis presented in Section 2.7, three ESBs were selected, and these are JBoss, Mule and ServiceMix ESBs. This section provides an overview of these ESBs in terms of their architectures and deployment models.

2.8.1 JBoss ESB

JBoss ESB is an open source ESB product from JBoss. It enables the integration of disparate systems by abstracting the differences between these systems and treats each as logical services on the ESB. JBoss ESB is based on four components which include (Conner *et al.*, 2012):

- Message Filtering and Message Listener route messages inside the ESB to the endpoint service or component. Message listener act as inbound message router which listen for messages in external channels. Upon receiving message, it is passed to message processing pipeline to evaluate message and determine its endpoint. Message Filtering filters out messages and route them to their destination endpoint.
- Data Transformation: Smooks action processors are used for converting data form one format to another within the ESB.
- Content-based routing: to achieve content-based routing inside JBoss ESB, there are several techniques available and they include XPath, Drools, Smooks and Regex.
- Message Repository: inside the JBoss ESB there is a registry for storing services using their EndPoints References (ERPs) to differentiate them.

2.8.1.1 Architecture

The architecture of JBoss ESB is shown in Figure 2.8. As part of trying to keep up with SOA principles, everything inside JBoss ESB is considered as either a service or a message (Conner*et al.*, 2012; JBoss Community, n.d).

Services– Services are the most important part of JBoss ESB they encapsulate business functions or point of integration with other systems like legacy systems. Inside JBoss ESB services are defined as a sequence of Listeners and Actions

- **Actions** are defined by java classes which are called one after the other for each processed message. For example an action can be sending messages to the content based router's class. The lists of actions are called Action Pipeline.
- **Listeners** are used to define input binding of the service. An example of Listeners can be http gateway or jms listeners. JBoss ESB define two kinds of Listeners which are Gateway Listeners and ESB aware Listeners.

Gateway Listeners are used for creating "Gateway" Endpoints. External consumers use these endpoints. These Listeners wrap messages coming from the outside of the ESB into an ESB aware message. After the message has been normalized it is then sent to Service's Action Pipeline for execution.

ESB Aware Listeners are used to create endpoints that exchange ESB Messages between the ESB Aware components inside the bus. These Endpoints are so called "ESB Aware Endpoints"

In addition to Actions and Listeners, Service name and category are other important attributes used for registering services into Service Repository. The knowledge of service name and category helps in the process of invoking the registered service.

Service configuration is done in an xml file called jboss-esb.xml. The file contains list of service definition that will be then deployed into an ESB.

Messages: Messages play an important part in communication inside JBoss ESB. Service clients and providers interact by exchanging a number of Messages. Message structure of messages inside JBoss ESB is based on SOAP message and extended by other fields to suit ESB.

- **Message Header** is mainly used for containing addressing and routing information for the message. Addressing inside JBoss ESB is based on WS-Addressing standard defined by W3C commit
- **Message Body:** this part of the message contains the actual message payload. Arbitrary number of Objects of any type can be contained inside the payload.
- Context contain information related to session, for example this can be transactional or security context
- **Attachment** can contain other payloads that are not inside the body, these payloads are contained in a form of attachments. Attachments can be any file type such as picture, document, zip package, etc.
- **Properties:** additional message meta-data are defined and contained in the message property. Similar to attachments, properties are treated the same way as message body.
- **Fault:**all the error information are contained in the fault field

Message Exchange Patterns and Response Handling

One-way and request-reply are the two Message Exchange Patterns supported by JBoss ESB. The simplest message pattern to manage is One-way communication message which is implicit for some contained class. For example, JMSRouter Action sends only one-way JMS message and it cannot be forced to receive response unless it's also a JMS response in a queue. Using request-reply communication style, each thread creates a temporary response destination that is automatically set in header of ESB message.

2.8.1.2 Deployment Model for JBoss ESB

JBoss ESB is integrated with JBoss server for runtime environment. A new project or service is deployed via the so called hot-deployment or by using the administration console via the web. Hot-deployment allows ESB projects to be deployed in the server while it's up and running.

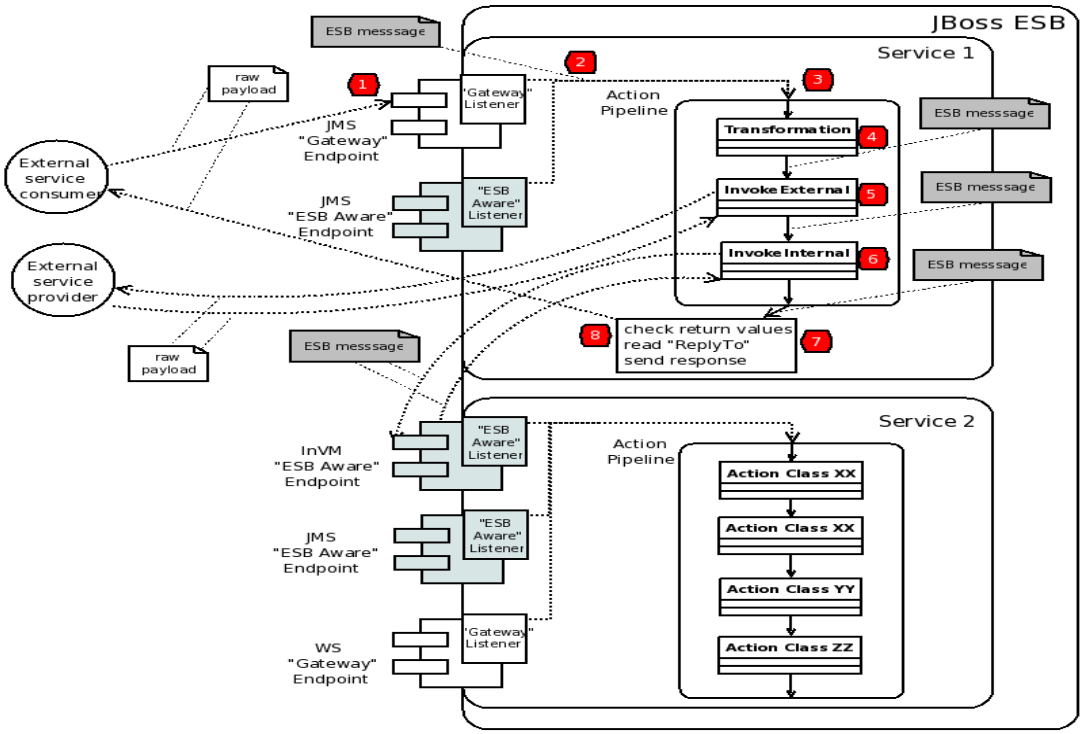

Figure 2.8: JBoss ESB Architecture using example (JBoss Community, n.d)

2.8.2 Mule ESB

Mule ESB is a lightweight java based open source ESB from MuleSoft Company. Mule service container is based on Universal Message Objects (UMOs) which are highly distributable object broker. It comes in two different versions which entail Enterprise and Community Edition. Their differences can be found in MuleSoft website (MuleSoft,n.d). This work focuses on the Community Edition.

2.8.2.1 Mule Architecture

Mule uses the concept of message flow which describes the message as it goes through a number of Mule components. Mule's Architecture consists of three layers, namely Application layer, Integration layer and Transport layer as depicted in Figure 2.9

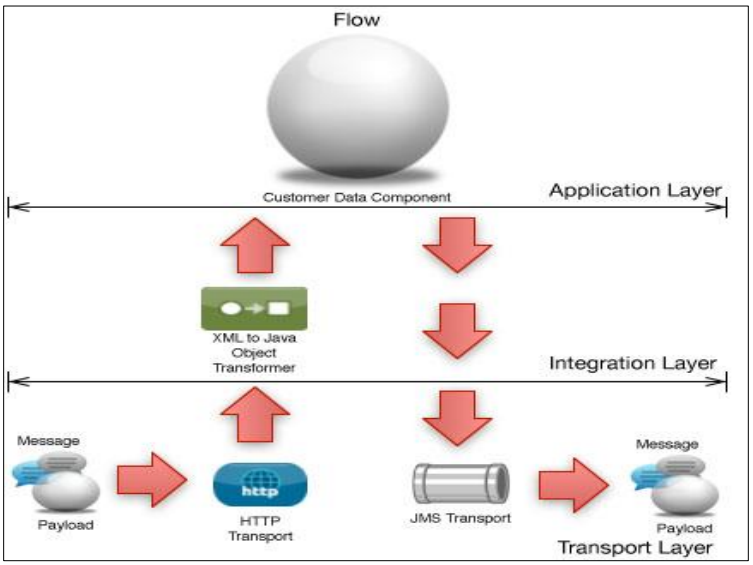

Figure 2.9: Mule ESB Architecture (MuleSoft, n.d)

Application Layer consists of back-end data component ranging from simple POJOs to complex web services that define business logic. These components do not need to include any Mule specific code. All the configurations needed are contained by XML file called mule-config.xml

which configures the service that wrap service component. Mule configuration file can be divided into three sections which are Mule Instance, Flow and Service. Mule Instance allows setting the global configuration options such JMS queue profile and error handling mechanism that is used by all services throughout the flow or service model. Flow enables to configure how messages are handled specifically how they are routed, filtered, and transformed between service components. Messages are sequentially processed by Mule components as defined in the flow. Lastly, Mule Services are the sectors where all services such as routers, filters and transformers are defined. Message routing in Mule is based on Enterprise Integration Pattern (EIP) as described in (Hohpe and Woolf, 2004). Most importantly Mule Service configures the service endpoint where the service will receive messages and outbound component where service will relay message next.

In the middle of Mule's architecture is the **Integration Layer**. As a part of supporting loose coupling SOA principle, this layer decouples services from underlying communication protocols to allow applications to communicate regardless of their heterogeneous protocols. To achieve full loose coupling this layer allows data transformed from one format to another. During data exchange transformers are the key components used. Mule defines a number of default transformers and they include XML-to-Object, JAXB XML-to-Object, XSLT, XQuery, and XPath to name a few. Custom transformers can also be defined to extend the existing default transformers when needed. This can be done using simple Java class.

The third layer and the last layer is **Transport Layer**. In this layer Mule define transport in which messages are received and dispatched on different protocol connectors. Connectors contain configuration and state of transport. Transports are mainly responsible for carrying messages from application to application. Mule support a wide variety of transport protocols out

of the box, these protocols includes HTTP, TCP, JMS, File, etc. As a part of integrating everything, Mule allows custom transport to be developed if the need arise in particular to support legacy systems.

2.8.2.2 Deployment Model for Mule

Mule deployment model allows managing application lifecycle and also enables multiple applications to run independent of each other within the same Mule container. The deployment model has hot deployed for new applications to be added and reloaded without having to restart Mule. In addition applications are designated to start with the server on startup (MuleSoft, n.d).

2.8.3 Apache ServiceMix ESB

Apache ServiceMix ESB is the integration platform that implements Java Business Integration standard (Ten-Hove and Walker, 2005). In general ServiceMix combines features offered by Apache ActiveMQ, Apache Camel, Apache CXF, Apache ODE (Orchestration Director Engine), and Apache Karaf into a powerful runtime integration platform. ActiveMQ provides reliable messaging and clustering of different components to improve fault tolerance and support failover mechanism. Apache Camel empowers definition of routing and mediation rules in a wide range of domain-specific language including Java. In addition Camel is used support Enterprise Integration Patterns (EIP) and different messaging models. ServiceMix support both traditional Web Services (WS) and RESTful (Representational State Transfer) web services through the use of Apache CXF. Long and short running processes defined using Web Service Business Process Execution Language (WS-BPEL) standard are executed using Apache ODE orchestration engine. Different services are orchestrated in a form of business process flow.

2.8.3.1 Architecture

Figure 2.10 below shows the basic architecture of ServiceMix ESB (Apache Software, n.d). As previously mentioned that ServiceMix is based on JBI specification therefore it's made up of three component. These components are Binding Component (BC), Normalized Message Router (NMR) and Service Engine (SE). On the Figure 2.10, the top layer shows examples of SE. SE is part of ServiceMix that implements business logic, transformation and routing services needed for application integration. At the middle of architecture is NMR which is responsible for facilitating communication between SE and BC. Messages passed in this layer are strictly XML massages. At the bottom is the BC which enables external components to have access to ServiceMix. Examples of BC are also shown in Figure 2.10and they are used for applications that communicate using SOAP and files. In addition Legacy applications are also offered protocols to interact with the ESB.

2.8.3.2 Deployment Model for ServiceMix

The new business logic in ServiceMix is made up of three components, namely SE, BC and service assembly. Service assembly is basically a collection of service units (describe SE and BC) packaged in a zip folder the can be hot deployed into an ESB. Hot deployment allows the flexibility to add new application or service to the running ESB server.

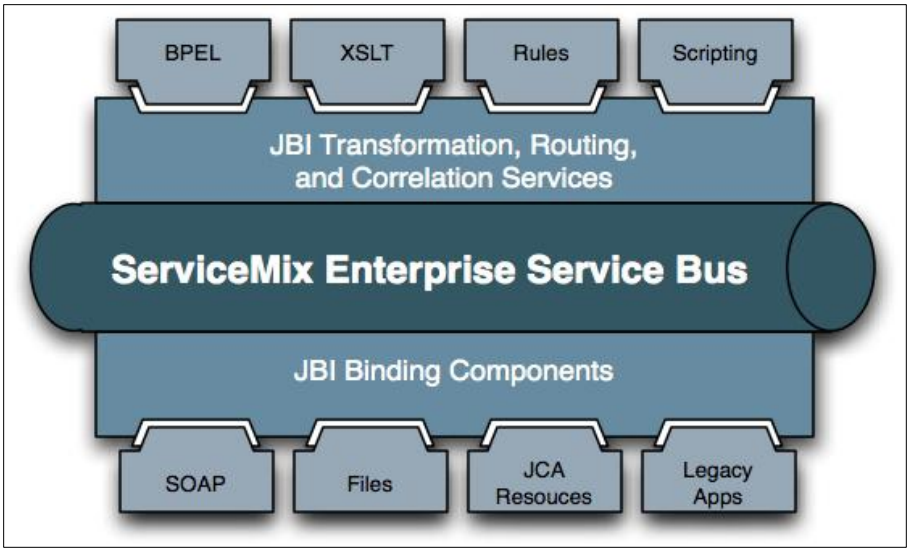

Figure 2.10: Apache ServiceMix ESB Architecture

2.9 Chapter Summary

This chapter introduced the GUISET concept and its integration requirements, thereby answering our first research question (as introduced in Chapter One, Section 1.2) which seeks to find out which integration capabilities will GUISET needs. Then it further presented an overview of the concept of enterprise integration and various approaches that have been adopted towards systems integration. Among these approaches Enterprise Service Bus is currently the state-of-art and it's widely adopted by enterprises worldwide (Vollmer *et al.*, 2011) (Issarny *et al.*, 2011) (Ortiz, 2007). The popularity of ESB is due to the fact that it is based on open standards and it paves a way for SOA. This chapter also introduced SOA and web service standards. In addition, a general evaluation of the top ranked ESBs was presented with an aim of selecting the ESBs that would be evaluated against GUISET integration requirement. From this general evaluation JBoss, ServiceMix and Mule ESBs were selected and the overview of the selected ESB platform was also provided. The next chapter critically analyzes existing work that has been done to evaluate ESBs. The analysis was further restricted to the work done related to evaluating GUISET integration requirements.

CHAPTER THREE

THEORETICAL EVALUATION OF ESBs

3.1 Introduction

Despite the major success of ESB technology as an integration solution for an Enterprise's IT resources, selecting a single ESB that optimally supports the integration requirements of a given environment is a challenging task. This is due to conflicting criteria that need to be considered in selecting an ESB to be used to best meet given integration requirements. Some research efforts (e.g. Siddiqui *et al.,* 2011; Vollmer *et al.,* 2011; Ahuja and Patel, 2011) have been directed towards finding ESB solutions that best fits a given environment. Such research efforts have produced sound methodologies that can be used in this endeavor, which the research reported in this dissertation builds upon. However, these efforts assume that there exist a single ESB that best meet all the integration requirements of a given environment. Contrary to this assumption, large SOA environments have a wide variety of integration needs that cannot be best meet by a single ESB. This dissertation argues that instead of compromising to a single ESB that best meet the conflicting criteria, multiple ESBs can be used in an integration solution with each ESB responsible for the capability it best supports.

To enable integration of multiple ESBs, this study firstly investigated which ESB has the best support for each GUISET integration requirements. GUISET integration requirements were presented in Chapter Two and they have support for high availability, data transformation, intelligent message routing, service composition and dynamic service discovery. This chapter presents ESB evaluation against the above mentioned GUISET integration requirements. The aim was to determine which ESB has the best support for which GUISET integration requirement. This was used to ensure that when integrating multiple ESBs, each ESB will be

used for the requirement it best supports. Section 3.1 establishes background for our evaluation by presenting review of the work done to evaluate ESBs using general criteria. The review of the work that has been done to evaluate the integration capabilities needed by GUISET is presented in Section 3.3. Section 3.4 presents the analysis of ESBs towards their support for GUISET integration requirements. The discussion on the analysis results is presented in Section 3.5. Finally, Section 3.6 provides the chapter summary.

3.2 General ESB Evaluations

There exist a substantial amount of research work that can be utilized when it comes to evaluating existing ESB solutions according to particular criteria and comparisons. One of the most important evaluations was done by Forrester research (Vollmer *et al.,* 2011). In this work both open source and commercial ESB products were evaluated against 109 criteria which were divided into three major groups that are "current offering" (capabilities of the product including architecture, orchestration, mediation, and routing), "strategy" (in terms of vendor cost, strategy and customer reference) and "market presence" (revenue, new customers, etc.). This evaluation considered a number of factors that can be used to assist in selecting ESB product to use.

Works that are closely related to the research reported in this dissertation are the evaluations that concerns with comparing performances of core ESB capabilities. These capabilities include high availability, data transformation, and content-based routing (Kruessmann *et al.*, 2009; Ahlberg *et al.*, 2010; Ahuja and Patel, 2011).

3.3 Core GUISET ESB Evaluation

ESBs share the same philosophy as Enterprise Application Integration (EAI) which is to provide integration of heterogeneous applications. ESB has been popular integration solution due to its

capability of integrating infrastructure where the concept of Web Service and SOA coalesce (Papazoglou and Heuvei, 2007). An ESB is the middleware glue that holds an SOA together and enables communication between web-based enterprise applications. Even though ESB concept is technically not a completely new concept, the drastically increasing adoption of SOA in the industry raises new engineering challenges that requires research to provide some solutions. These challenges include the performance evaluation of SOA systems and enabling infrastructures. This work will focus on performance evaluation of ESBs. Performance comparison of ESB products has been the focus of many researchers as it assists in ESB selection. However, testing the end-to-end performance of any SOA systems including ESB is often a difficult task due to the barriers that exists in this environment. These barriers include organization boundaries, presence of shared resources and services with other organizations, security requirements and the fact that load testing can be perceived as a denial of service attack to the system. These barriers are among factors that impose restrictions on performance testing of SOA systems. On the other hand, integration testing on the production ready system can be inevitably too late and expensive to completely change software architecture to address performance and scalability problem that might arise. It is, therefore, valuable to predict the performance implications of architectural alternatives for SOA as early as possible during development lifecycle. The method that allows performance and capacity of the ESB to be tested at planning stage before the actual deployment was proposed in Ueno and Tatsuburi (2006). The method is based on capacity-planning methodology (Menasce and Almeida, 2002) and it was intended to assist designers to be able to configure lightweight environment for testing an ESB capacity and performance at an early stage before the actual deployment. This work is relevant to ours as it justifies and proves that lightweight environments configured with limited hardware

resources for ESB assessment achieve equivalent results to that of production systems used in the real world environments. Furthermore, another approach towards early performance prediction of SOA applications is proposed in Brebner (2009). This approach used model-driven approach to automatically generate a runtime model for execution of composite services and workflows. The discrete-event simulation engine dynamically captures performance of each service that participates in the composition. Therefore, this allows performance adjustments to be done on a specific service rather than the whole composition. The famous SOA/ESB example of LoanBroker application (Hohpe and Woolf, 2004) and Mule ESB was used to test the method. Brebner's approach showed that the performance of SOA applications can be easily modified and maintained. The above mentioned work does not provide performance comparisons of different ESB products, but they propose the valuable methods for ESB evaluation. Given the particular requirements of SOA application and environment, organizations would like to evaluate different ESBs. This evaluation can help an organization to select an ESB that best supports the desired integration requirements. A lot of research has been done to evaluate performance of ESB products based on a given scenario. This work discusses research efforts related to the evaluation of GUISET integration requirements as they are highlighted in Chapter Two. These requirements are High availability, Data Transformation, Intelligent Message Routing, Service Composition and Dynamic Service Discovery.

3.3.1 ESB High Availability

High availability characterizes an ESB that is designed to avoid the loss of services by managing or reducing failures. Most ESBs use failover mechanisms to ensure high availability. Failover mechanism requires more than one ESB instance to be up and running so that when the primary instance fails the secondary instance takes over and resumes where the primary instance left. The

major issue is to disseminate state information among the cluster. Staged Event Driven Architecture (SEDA) service event and In-memory message queues help to solve this issue by storing the transient state information and sharing among running instances (MuleSoft, 2012). In this direction the work done by Siddiqui *et al.,* (2011); Kruessmann *et al.*,(2009) and Ahlberg *et al.*,(2010) evaluate high availability. These evaluations use Java Messaging System (JMS) to support failover that ensure high availability of both ESB and services deployed in it. JMS allows clients to send message to a list of broker connection URLs so that when one broker fails the other one takes over. Failover capabilities were simulated by killing the server process, unplugging network cable and killing the master. Some ESBs are not too dependent on JMS implementation, for example Mule does not include any JMS on the product. However APIs and mechanisms are available to integrate any JMS compliant product. On the contrary, some ESBs have strong support for JMS, for example ServiceMix is distributed with ActiveMQ while JBoss is shipped with JBossMQ. Regarding high availability the ESBs with strong support for JMS are more favored (Kruessmann *et al.,* 2009).

3.3.2 Data Transformation

Another important capability of ESB is to ensure that different application understands each other regardless of their underlying technology. This requires that data from one system be transformed from one encoding to the other and this is done by an ESB capability known as data transformation. Data transformation is the technique used to change data from one format to another as required by destination application (Genender, 2006). Most ESBs support XML as native format for data flowing across the bus. Extensible Stylesheet Language Transformations (XSLT) is the most used method for data transformation. The performance of XSLT in transforming XML messages has been evaluated by a number of authors including

AdroitLogic,(2012); Ahuja and Patel,(2011); Perera,(2008);Desmet *et al.,* (2007). In all the afore-mentioned work XSLT was applied as a proxy between two communicating applications. The performance metrics used for evaluation included response time, throughput and load handling. Data transformation contributes to lose coupling between interacting applications because the sender needs not to worry about the format expected in the destination. Mule's transformer allows transformation objects to be pooled for better performance. This means transformation context properties can be set on the transformer and pooled from message using expression evaluator. This flexibility achieves better performance.

3.3.3 Intelligent Message Routing (Content-based routing)

Routing capability allows ESB to route messages to the appropriate destination application. ESBs implement content-based routing which is a facility that allows a message to be received, analyzed and based on its content the destination application is determined (Ziyaeva *et al.*, 2008). Content-based routing is typically used in practice to route requests messages to different destination endpoints or to handle them differently at the ESB based on their message attributes. There has been some research efforts directed at evaluating the performance of ESBs in the context of content-based routing techniques (Perera, 2012; Ahuja and Patel, 2011; Kusak, 2010). In the afore-mentioned work, an ESB performs XPath evaluation over a SOAP message to determine the message destination. In these evaluations, the performance metrics considered were response time, throughput and load handling. Load-handling was characterized as a single client sending varying sizes of message payloads. The results of these evaluations showed that on content-based routing Mule performs better than ServiceMix, Fuse, JBoss, and Open ESB. This is attributed to its powerful expression-filter that allows enhanced configuration of XPath and Groovy.

3.3.4 Dynamic Service Discovery

In real world scenarios, large organizations maintain a huge number of services that implements some business logics that is needed for the day-to-day running of the business. These services need to be managed in a most efficient manner to ensure that they can be automatically discovered when needed by other services or business processes. Efforts have been made to enable dynamic service discovery for services integrated through an ESB. Chen *et al.,* (2008) proposed an adaptive service bus that is able to support dynamic change of business rules at runtime to avoid costly unexpected shutdown of applications when faults happen. In this approach, a service router component of a dynamic service bus is responsible for discovering and selecting the service to be used at runtime. The targeted service selection is based on the ranking of different services which uses information such as expected availability and execution time. Dynamic Routing in Enterprise Service Bus (DRESR) was proposed by Bai *et al.,* (2007) to allow service routing table inside the ESB to be changed at runtime. DRESR used abstract service name to define routing path in which at runtime the real services are instantiated by replacing the abstract service names with the real URIs. In addition DRESR supports service selection preferences such as response time.

Wu *et al.,* (2008) also proposed another approach towards achieving dynamic service routing. This approach was enhanced with reliability of sending requests by redirecting the requests to another service when the response from the previous service was not received within suitable time. Context of application information related to the request was used for discovering list of targeted services. Moreover, Jongtaveesataporn and Takada (2010) enhanced dynamic service selection by load balancing. Service types were used to categories services with the same functionalities. The service was dynamically selected based on the server status and load which

was constantly monitored by load monitor. Yu and Yan, (2011) proposed another approach that integrates ESB and UDDI registry to allow service endpoint to be obtained dynamically when needed by to route messages in the ESB. Although the above mentioned approaches provided the remarkable effort towards dynamic service routing, but they do not present any performance evaluation of different ESBs when considering dynamic service discovery.

3.3.5 Service Composition

In addition to UDDI support, in order to achieve a certain business objective, it is often that multiple services are composed to create a business process. Composition of services can be achieved either through service orchestration or choreography. Most ESBs provide support for BPEL which is the orchestration language for defining processes. The work by (Garcia-Jimenez *et al.,* 2010) showed the performance evaluation of ESBs invoking the existing external BPEL process.

3.4 Analysis Methodology

Making decision on competing ideas/solution/artifacts that have multiple conflicting criteria is a challenging task that has drawn attention in literature from multiple disciplines. Solutions for such task fall under a set of methods known as Multi Criteria Decision Analysis (MCDA). MCDA has been acknowledged as an important tool in environmental decision-making for formalizing and addressing the problem of competing decision objectives. In general, the goal of MCDA is to determine a preference ordering among a number of available options using multiple criteria (Steele *et al.,* 2009). MCDA methods include Analytical Hierarchy Process (AHP), Bayesian Analysis (BA), and Multi-attribute Utility Theory (MAUT) method (Cho, 2003). However, AHP since its invention has been the most widely used MCDA method by decision makers and researchers. Outstanding results have been published on AHP application in

different fields including software engineering (Triantaphyllou and Mann, 1995), agriculture (Alphonce, 1996), and project management (Al-Harbi, 2001). AHP allows a decision to be made using either qualitative or quantitative data. AHP use pair-wise comparison to determine trade-offs among criteria and the ability to calculate the degree of consistency (or inconsistency) of judgments in each step. The possibility of applying AHP to evaluate some ESB capabilities is presented by Siddiqui *et al.,* (2011). Siddiqui *et al.,* (2011) used Interoperability, Information Security and High Availability as their criteria. Taking inspiration from the success of using AHP method in making a decision by considering multiple criteria, the work reported in this dissertation used AHP to resolve issues of conflicting ESB selection criteria. The AHP process used for the analysis in this study is known as Pair-wise comparison (Saaty and Shih, 2009). The pair-wise comparison allows a decision to be made considering multiple criteria. These criteria are ranked according to their importance. Using pair-wise comparison, an enhanced decision that is acceptable and un-contradictory can be achieved.

3.4.1 Pair-wise comparison process

There are three hierarchical steps to be observed during the pair-wise comparison process. These steps are ‘Goal’, ‘Criteria’ and ‘Alternatives’ as illustrated in Figure 3.1. In this study, the goal is ‘Analysis of ESBs for GUISET’, the criteria used for this analysis are ‘High Availability’, ‘Content-Based Routing’, ‘Data Transformation’, ‘Service Orchestration’ and ‘Dynamic Service Discovery’, the alternatives ESBs considered are ‘Mule ESB’, ‘ServiceMix’ and ‘JBoss ESB’. Three pairs of the AHP pair-wise comparisons were made. These pairs were Mule was compared with ServiceMix, secondly Mule was compared with JBoss and lastly ServiceMix was compared with JBoss.

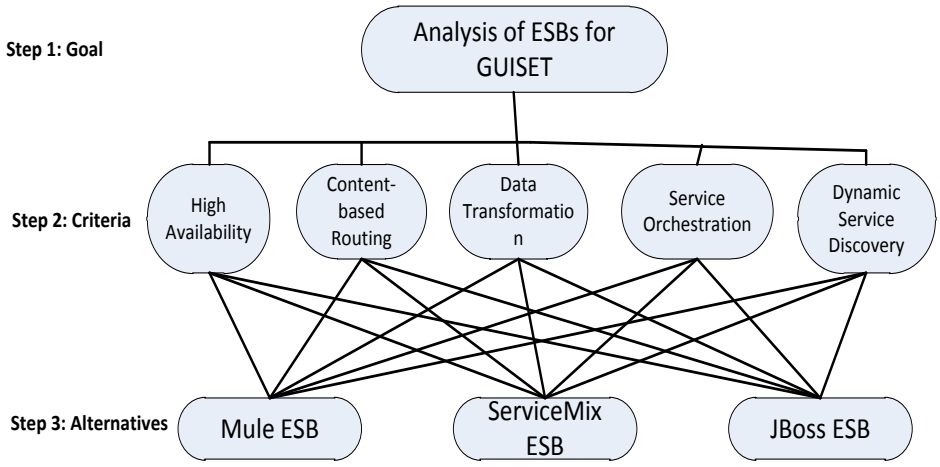

Figure 3.1: AHP steps for Analysis of ESBs for GUISET

The pairwise comparison was used to determine the relative importance of each alternative ESB in terms of each criterion. The priority assignments and judgments on ESBs are presented in Section 3.4.2. The priority assignments were based on the nine-point scale proposed by Saaty, (2007). The scale is as described in Table 3.1 below. The overall rankings of both selection criteria and ESBs are given in Section 3.4.3 and Section 3.4.4, respectively.

Table 3.1: Intensity Scale and Definition

Intensity importance	**Definition**
1	Identical
3	Considerable in favor
5	Strongly in favor
7	Very strongly in favor
9	Acute favor
2, 4, 6, and 8	Intermediate values between judgments, used when compromise is needed

Table 3.2: ESB technologies towards supporting GUISET integration requirements

ESB(s)	ESB products effort towards support of GUISET integration requirements
Mule ESB	**High Availability** Mule support clustered ESB instances, which uses StagedEvent Driven Architecture (SEDA) service and in-memory message queues. High availability of services on transactional transports like JMS, WebSphere MQ, and JDBC. However, for non-transactional transport like HTTP (including CXF web services), File and FTP Mule uses reliable patterns called reliable requisition flow (MuleSoft, n.d). **Message and Data Transformation** Messages exchanged in mule can be any format, Mule support JSON, Scripting, Encryption and XML based transformation using XSLT, XStream, JAXB binding framework, XQuery, XmlToDom, XPath expression using JAXP and JXPath. Custom transformers are offered via Apache velocity Engine. **Content-based Routing** For content-based routing Mule use filters, which leverages Mule expression to all configuration of XPath, Groovy and OGNL based filters. XPath filters are implemented with Apache Commons JXPath Library. Payload type, Regular expression and wild card filter is used to determine endpoint service. Mule's ChoiceRouter choose one and only on endpoint route that matches those in the filter. **Service Orchestration** Mule support JPBM process engine and JPDL suite distribution. Mule support invocation of services hosted in any BPEL engine like Oracle SOA BPEL engine using standard SOAP endpoint (web services) and JMS. **Dynamic Service Discovery** Galaxy service registry is used to store and discover service related metadata. Among features provided by searching capabilities using SQL like queries and support for Atom Publishing Protocol (APP). For view, publish, and subscriber Galaxy use Queues, web interface/HTTP, open search, custom query language, XQuery, XPath and Groovy.
ServiceMix ESB	**High Availability** Support WS-Reliable Message standard for CXF binding component which are non-transactional. Non-persistent messages in-memory SEDA flow is used. Alternatively, persistent message on a transactional transport Active MQ can be used including JDBC clustering. Active and standby clustering ensures high availability(Apache Software, n.d). **Message and Data Transformation** Since all messages exchanged in NMR are XML, so Saxon service engine is used for

	transformation, Saxon is based on XSLT style sheet. XQuery based transformation is also supported. Additional message can be transformed via script, Java class and BPEL. **Content-based routing** This feature is supported using XPath expression on a normalized XML content. Also EIP and Camel service engine. Rules-driven routing using drools and script-driven routing using servicemix-script service engine. **Service Orchestration** Strong support for BPEL specification using Apache ODE as an external service engine. Engine provides a drop-in JBI to execute composite services inside ServiceMix. Alternatively Routing and transformation mechanisms can be used with support of EIP (for individual service connection) and Script or Java beans **Dynamic Service Discovery** JBI provides a very basic registry which map services to WSDL and can be accessed via JBI API (with a provided component Context) or remotely via JMX. Apache jUDDI can be used as an external service to provide more sophisticated service discovery capabilities.
	High Availability
JBoss ESB	High availability is supported via JBoss MQ (Re-delivery queue), JBoss massaging and JBoss Messaging Core for transactional, reliable transport system (JBoss Community,n.d). **Message and Data Transformation** Smooks is the default transformation engine, but XML-based transformation engine XSLT is also supported. Custom transformation is supported using Action Processor data transformation. **Content-based routing** Provide support for JBoss Drools rule engine which is a complete enterprise platform for rule based application development, alternatively simpler approach include XPath custom rule and Regex Content based routing **Service Orchestration** Strong support for jBPM which is a default BPM engine for orchestrating services. Alternative from jBPM, ActiveBPEL andRiftsaw BPEL engine can be used. Also support for WS-BPEL via Web service component. **Dynamic Service Discovery** By default JBoss ESB uses JAX-R implementation (Scout) and UDDI (jUDDI) for service registry and discovery. Registry stores EndPoint References (ERPs) for services deployed and its automatically updated when new service start

3.4.2 Priority Assignment and Judgments

The weights were assigned using two sources of information; vendor documentation on the ESBs and publicly available evaluations of the ESBs published by independent entities. At first we reviewed the documentation of each ESB products as published by vendors (MuleSoft,n.d), (Apache Software,n.d), and(JBoss Community,n.d). Table 3-2 shows summaries of the efforts in terms of technologies made by each ESB (Mule, ServiceMix and JBoss ESB) in support of GUISET integration requirements. According to this review it was noted that all ESBs use different approaches in providing each capability. However, some technologies are widely used and they keep appearing like transformation using XSLT and content based routing using XPath expressions. The information in the vendors' documentations served as the background knowledge. We then looked at the published and publicly available articles on ESB evaluations with the aim of verifying the claims made by vendors on the product documentations. The articles were selected on the basis that they provide either empirical or qualitative analysis of the ESB products that are under review. Moreover, the articles were evaluating high availability, content-based routing, data transformation, service orchestration and dynamic service discovery capabilities of different ESBs as presented in Section 3.3. Table 3.3, Table 3.4 and Table 3.6 represents 3 X 3 matrices for the corresponding judgments in each decision criteria (GUISET integration requirements). Alternatives listed on the left are one-by-one compared with each alternative listed on top so as to determine which ESB is more preferred than the other.

Table 3.3: Pairwise comparison matrix for High Availability (HA)

High Availability	Mule	Service Mix	JBoss
Mule	1	1/3	3
SMX	3	1	5
JB	1/3	1/5	1

A normalized pairwise comparison matrix can be obtained bydividing each element of the matrix by its column total. For example, value 0.2308 in Table 3.4 was obtained by dividing 1 (from Table 3.3) with the sum of a column items *(1 +3 +1/3)* from Table 3.3. Priority vector (Eigenvector) in Table 3.4 was obtained by finding the row averages. For example, the priority vector of Mule with respect to the criterion '*High Availability*' in Table 3.4 was calculated by dividing sum of the rows *(0.2308 + 0.2174 + 0.3333)* with number of columns (alternative ESBs), i.e. 3 to obtain 0.2605.

Table 3.4: Normalized table for *High availability*

High Availability	Mule	Service Mix	JBoss	Priority Vector
Mule	0.2308	0.2174	0.3333	0.2605
SMX	0.6923	0.6522	0.5556	0.6333
JB	0.0769	0.0222	0.1111	0.1062

λ_{max} = 3.0387, CI = 0.0194, RI = 0.58, CR = 0.0334 < 0.1 OK

Now having the pairwise comparisons which are given in Table 3.4, the consistency ratio was determined by using the eigenvalue λ_{max}.

Eigenvalue was calculated by first finding the weighted sum average of the given matrix. For example considering judgments for criterion "*High Availability*", the weighted sum average is given as:

For example Consistency Ratio (CR) for criterion '*High Availability'* can be obtained as follows;

$$0.2605\begin{pmatrix}1\\3\\1/3\end{pmatrix} + 0.6333\begin{pmatrix}1/3\\1\\1/5\end{pmatrix} + 0.1062\begin{pmatrix}3\\5\\1\end{pmatrix} = \begin{pmatrix}0.7902\\1.9458\\0.3197\end{pmatrix} \quad \begin{matrix}\text{weighted}\\\text{sum}\\\text{Matrix}\end{matrix} \qquad \textbf{(1)}$$

Then, having weighted sum matrix, we calculated *Eigen value* by dividing all elements of the weighted sum matrices by their respective priority vector elements. We then found its average as follows:

λ_{max}= [(0.7902/ 0.2605) + (1.9458/ 0.6333) + (0.3197/ 0.1062)]

3

= **3.0387** **(2)**

Now, we find the *Consistency Index* (CI) given by the formula

CI = (λmax – n) / (n – 1)

- *n*, is the matrix size
- λ_{max} is the Eigen value as calculated above

CI = (3.0387 -3) / (3 -1) = **0.0194** **(3)**

The judgment consistency can be checked by taking the *Consistency Ratio* (CR) of CI with *Average Random Consistency* (RI) given in Table 3-5 below. The CR is acceptable if it does not exceed 0.10, if it does there is a need to review and improve judgments.

Table 3.5: Average Random Consistency (Al-Harbi, 2001)

Size of matrix	1	2	3	4	5	6	7	8	9	10
Random consistency	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

Finally, the appropriate random consistency value was selected. Using the matrix size of 3 from Table 3-3, the random consistency value RI = 0.58 (from Table 3.5 above). The consistency ratio CR was calculated as follows:

CR = CI/RI = 0.0194/0.58 = **0.0334** **(4)**

Complete judgments for other four remaining criteria

Table 3-6 shows the judgments for content-based routing, dynamic service discovery, data transformation and service orchestration.

Table 3.6: Pairwise Comparison matrix for the remaining criteria

Content-based Routing (CBR)

Priority Vector

Mule	1	5	7	0.7235
SMX	1/5	1	3	0.1932
JB	1/7	1/3	1	0.0833

λ_{max} = 3.0623, CI = 0.0312, RI = 0.58, CR = 0.053 < 0.1 OK

Dynamic Service Discovery (DSD)

Priority Vector

Mule	1	3	1/3	0.2431
SMX	1/3	1	1/7	0.0882
JB	3	7	1	0.6687

λ_{max} = 3.0072, CI = 0.036, RI = 0.58, CR = 0.0621 < 0.1 OK

Data Transformation (DT)

Priority Vector

Mule	1	7	5	0.7235
SMX	1/7	1	1/3	0.0833
JB	1/5	3	1	0.1932

λ_{max} = 3.0623, CI = 0.0312, RI = 0.58, CR = 0.053 < 0.1 OK

Service Orchestration (SO)

Priority Vector

Mule	1	1/2	3	0.3325
SMX	2	1	3	0.5278
JB	1/3	1/3	1	0.1397

λ_{max} = 3.0387, CI = 0.01935, RI =.58, CR = 0.0334 < 0.1 OK

3.4.3 The Overall Criteria Rankings

In addition to the pairwise-wise comparison for the decision alternatives, the same procedure was also used to set the priorities for all our five criteria so as to find the importance of each criteria in contributing to the overall goal. Table 3.7 shows pairwise comparison matrix for the five criteria. Figure 3.2 graphically represents criteria rankings, with Dynamic Service Discovery and Service Orchestration being the most important criteria for GUISET, followed by High Availability, Content-based Routing and Data Transformation, respectively.

Table 3.7: Pairwise comparison matrix for all five criteria

Criteria	SO	HA	DSD	DT	CBR	Priority Vector
CBR	1	1/5	1/3	1/2	1/5	0.0621
DSD	5	1	3	2	1	0.3252
DT	3	1/3	1	1/2	1/3	0.1192
HA	2	1/2	2	1	1/2	0.1682
SO	5	1	3	2	1	0.3252

λ_{max} = 7.1511, CI = 0.5378, RI = 1.12, CR = 0.4802 < 0.1 OK

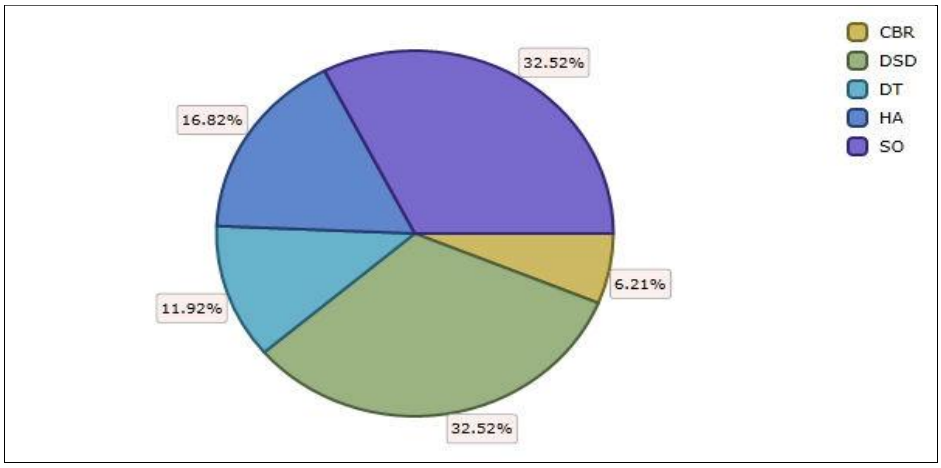

Figure 3-2: Average weights and Ranking for all criteria

3.4.4 The Overall ESB Ranking

Section 3.4.3 above presented the judgment of each ESB in a given criterion. Using these priority judgments, our goal was to rank the three alternative ESBs according to their support for GUISET integration. This section shows the ESB ranking according to their overall priorities. For example, overall priority for Mule ESB was obtained as follows:

Overall priority for Mule ESB

= 0.7235(0.0621) +0.2431(0.3252) + 0.7235(0.1192) +0.2605(0.1682) +0.3325 (0.3252)

= 0.3622

For other ESB, overall priorities are given in Table 3.8 below

Table 3.8: Overall ESB Rankings

Criterion Alternatives	CBR (0.0621)	DSD (0.3252)	DT (0.1192)	HA (0.1682)	SO (0.3252)	Overall Priority
Mule	0.7235	0.2431	0.7235	0.2605	0.3325	0.3622
SMX	0.1932	0.0882	0.0833	0.6333	0. 5278	0.3268
JB	0. 0833	0.6687	0.1932	0.1062	0.1397	0.3096

Therefore, based on the AHP analysis performed, Mule came first in the ranking followed by ServiceMix and JBoss ESB, respectively.

3.5 A discussion on Analysis of Results

According to the comparative analysis presented in Section 3.4, it can be observed that the presented ESB solutions are somehow similar because they all have some mechanism towards supporting the integration requirements needed by GUISET. However there is no one ESB solution that fits all for a dynamic environment like GUISET. For example scalability is an important requirement, so the distributed topology of Mule ESB allows it to be more scalable than other ESBs (Lima, 2011). In addition Mule allows transformation to be available as part of the enterprise services bus rather that keeping it inside the service components. This approach matches the concept of Aspect Oriented Programing (AOP) which makes Mule achieves more effective data transformation technique that results in better performance (Desmet *et al.*, 2007) (AdroitLogic, 2012). Moreover, Mule architecture is based on Enterprise Integration Pattern that allows it to have support for more enhanced message routing capability (Hohpe and Woolf, 2004). However Mule ESB does not have native UDDI support to enable dynamic service discovery. In addition, support for high availability of service is low because Mule architecture is not based on any JMS implementation.

On the other hand, ServiceMix has a strong support for JMS and its architecture is based on Apache ActiveMQ to enable the better availability of services and cluster instances for failover (Kruessmann *et al.*, 2009). Moreover, ServiceMix is shipped with Apache Orchestration Director Engine to support execution of BPEL processes. JBoss ESBs comes integrated with Java implementation of the Universal Description, Discovery, and Integration (jUDDI) registry to provide UDDI capabilities including publishing and discovering of services via an ESB (Conner *etal.*,2012). Once a service is requested by the consumer, JBoss ESB dynamically gets its address

to jUDDI. Since the basic information about services running in the ESB are stored in jUDDI server, service management become easier.

3.6 Chapter Summary

This chapter presented the work that was done to evaluate ESBs. The review was restricted to the work that evaluates ESB based on integration requirements for GUISET in order to provide the theoretical background of this work. Reviewing literature and presenting the results was not enough, so the further analysis was presented with an aim of determining which ESB has better support for which GUISET integration requirements. The analysis presented in this chapter provides the useful information that assisted in providing the answers to our second research question that seeks to investigate which ESB better support each GUISET integration requirement. However, the analysis was based on the qualitative data collected about each ESB. This data was reported in the literature.

Although works has been done to assist ESB selection process but there still a lot to be desired in particular when considering dynamic service discovery and service orchestration. There are few works that attempted to enable dynamic service routing by integrating ESB with UDDI (Yu and Yan, 2011). However, no comparison of the ESB solutions was mentioned. Moreover dynamic service discovery and selection was not addressed. The performance evaluation of ESBs that invokes external BPEL process was reported in (Garcia-Jimenez *et al.*, 2010). Nevertheless, there was no integration of ESB with BPEL engine to allow the execution of local business processes. Moreover the services were pre-defined at design which makes the service routing to be static. These points are worth mentioning because during our criteria ranking dynamic service discovery and service orchestration were ranked at the top as the most important requirements for GUISET integration. The rest of the work presented in this dissertation focused on these two

criteria. The importance of these integration requirements led to the need to empirically evaluate Mule, ServiceMix and JBoss ESB towards support of dynamic service discovery and service orchestration. This empirical evaluation is reported in the next chapter.

CHAPTER FOUR
EMPIRICAL EVALUATIONOF ESBs

4.1. Introduction

The literature reviewed in Chapter Three shows that the qualitative and empirical evaluations for high availability, data transformation and content-based routing has been an attraction to a number of researchers. ESB evaluation regarding service orchestration and dynamic service discovery has not been fully explored. This is mainly due to the fact that most ESBs do not provide these features out of the box. However, there exist add-ons to support these features (Mulik, 2009). Dynamic service discovery allows appropriate service to be determined at runtime. UDDI is the key enabling specification that can be used for dynamic service discovery and management. The use of BPEL standard for defining short and long business processes enable the existing enterprise functions to be composed and reused to build new business requirements. There are engines that have been implemented to execute BPEL processes. Most ESBs provides add-ons for these two features. Therefore, this chapter presents the empirical comparative evaluation of ServiceMix, Mule and JBoss regarding service orchestration and dynamic service discovery. More specifically, the following key questions were investigated:

- What is the effect of integrating ESB with BPEL engine on ESB performance?
- What is the effect of integrating ESB with UDDI on ESB performance?

The aim of this chapter is to provide answers to these question and select the ESB that perform better when given a service orchestration and dynamic service discovery. This chapter begins by an overview of ESB with service orchestration and ESB with UDDI in Section 4.2 and Section 4.3 respectively. The LoanBroker scenario is presented in Section 4.4. The ESBs use different approaches towards realizing integration capabilities, Section 4.5 present our integration model

design and the configurations of each ESB. Section 4.6 presents the ESB performance evaluation. Finally, Section 4.7presents the chapter summary.

4.2. ESB with Service Orchestration

The popularity of web service technology and the growing adoption of service-based application development models resulted in the possibility of having software applications that are built as a composition of existing services. Service composition is one of the most important principles of SOA. A composite service combines more than one service functionalities to achieve a new business requirement. The process of composing functionalities of different services is known as service composition. An instance of a composite service is often called business process. In SOA there exist two approaches towards service composition. These include service choreography and orchestration. However, service orchestration is the most supported approach by ESB. The popularity of service orchestration led to WS-BPEL standard being proposed and widely accepted as the language for defining business processes.

4.2.1 BPEL Engine and WS-BPEL

BPEL engine executes and provides runtime environment for business processes written in WS-BPEL standard. The engine coordinates invocation of web services by sending and receiving messages, handling data manipulation and error recovery as defined in the business process. BPEL engines support execution of both long and short living business processes. Business processes are defined using WS-BPEL standard which is a language that defines several constructs for writing business processes. Processes are defined by a set of basic control structures such as conditions, invoke web service element and receive messages form service. In order to invoke web service, a WSDL interface of a service is used.

There are quite a number on BPEL engines that are available and they include Apache ODE, JBoss Riftsaw, Open ESB BPEL Engine, OW2 Orchestra and Petals BPEL Engine. They are different in terms of their architecture but they share the same philosophy of executing WS-BPEL defined business processes.

The ESBs under review in this work has strong support for Apache ODE and JBoss Riftsaw. However, JBoss Riftsaw is a WS-BPEL engine optimized for JBoss application server container. JBoss Riftsaw is not a standalone BPEL engine; it's based on Apache ODE. Therefore Apache ODE was considered for service orchestration.

Apache ODE

Apache ODE is an engine which executes one or more business processes defined in WS-BPEL language. Apache ODE supports two communication layers, one based on Axis2 (specifically, Web Services http transport) and the other based on JBI standard. It has compatibility with BPEL4WS 2.0 which includes WS-HumanTask with Apache HISE.

4.3. ESB and UDDI

ESB supports systems communication by providing standard based platform for integrating heterogeneous applications. Inside the ESB service invocation and composition are implemented by routing messages to a specific destination endpoint configured in the configuration file. This is static service routing because destination services are not discovered to determine their status dynamically. The current ESB frameworks support only this static service routing which can result in invocation faults if a specified destination is unavailable. Moreover, in a dynamic environment like SOA, services are added and updated overtime to meet new requirements, so it becomes difficult to manage these services that might be provided by different systems that are

heterogeneous and distributed in nature. This section, therefore, presents the method for integrating ESB and UDDI service registry to enable dynamic service discovery and easy management of services inside the ESB.

4.3.1 The UDDI Registry

The UDDI is currently the most widely used service registry and it has been voted to be a mature service registry by OASIS (Longworth, 2005). UDDI is an effort done by the industry to provide a directory service for web services offered by different businesses. UDDI allows business providers to publish their services in the directory and enables business partners and consumers to discover and use those services. To make publishing, discovery and binding of services possible, UDDI specification provides structural templates for representing information about business entities, information about their services and the mechanisms to access them. One of the templates structure is the tModel. The tModel describes technical details about the service. Structural templates are facilitated by standards such as WSDL and SOAP to enable quick business-level and service-level discovery.There are several implementations of UDDI that are currently available and they include UDDI4J (UDDI for Java) and jUDDI (Java UDDI). These implementations make it easier to search or publish services without getting mired in the complexities of the UDDI API. UDDI4J is the java implementation of the UDDI that was originally developed by IBM. Apache jUDDI is another java UDDI registry implementation that provides toolkits for accessing and interacting with UDDI. In this work we have used apache jUDDI to assist ESB to support dynamic service discovery.

Apache jUDDI

Apache jUDDI can be deployed on Apache tomcat server and support different database management systems such as MySQL, Oracle and Derby. Derby is a default database that comes integrated with jUDDI and therefore it was used to avoid complexity in our implementation. Apache jUDDI support three APIs to interact with registry: publish API, search API and security policy API. Publish API provides set of functionalities for publishing new businesses and services. Search API provides functionalities for searching registered businesses and web services. Finally, security policy API ensures security for publish and search operation by granting authentication token to users performing operations in the registry.

4.3.2 Dynamic service discovery and Selection

Discovery in the UDDI is done using a set of search criteria for finding businesses and their services. Services can be searched by specifying business name, service name, service category and tModels. As the number of service providers advertising their services increase, the number of web services offering the same functionalities will also increase. This means that more and more services meet the requester's search criteria, so selecting only one service becomes a problem. An approach to useQoS attributes to differentiate between services that offer the same functionality was proposed and it's now widely adopted by many researchers (Zeng*et al.*, 2004; Khan *et al.,* 2010; Rajendran and Balasubramanie, 2009). QoS attributes define non-functional information about the service. These attributes include response time, availability, and reliability (Wu and Wu, 2010). To have web services that are QoS-aware there is a need to provide structure to allow service providers to publish their web services together with their QoS information. The UDDI specification does not include the structure for defining QoS information. However, Blum and Fred (2004) proposed that the QoS information of the web

service be stored in the UDDI tModel categorization. Many researchers have adopted this approach and have proved to be more efficient than extending UDDI data structure using UDDI's extensibility mechanism (Patil and Gopal, 2012, Tiwari *et al.*, 2012, and Khan*et al.*, 2010). Blum's approach uses *categoryBag* for defining QoS mechanisms. Inside *categoryBag* there is *keyedReference* that contains the *keyName*, *keyValue* and *tModelKey*. The *keyName* represent the name of the QoS attribute. The *keyValue* is the corresponding QoS value assigned to the attribute. The *tModelKey* is the unique identifier for the QoS information. This work considered only service availability as the QoS attribute for differentiating services with the same functionality. Service availability is defined in percentage with the value between 0 and 100. During dynamic service discovery the service availability value is obtained and compared with availability of other services. The service with high availability is selected for binding.

4.4. Motivating Scenario: Loan Broker Application

In order to demonstrate the applicability of service orchestration and dynamic service discovery, this section presents experimental prototype implementation of LoanBroker user case scenario.The Loan Broker application is a widely usedSOA/ESB application example, and is welldocumented in the Enterprise Integration Patterns book (Hohpe and Woolf, 2004). However, it is important to note that although Loan Broker scenario presents real world example of service orchestration, the concept in this work applies to a lot of service orchestration scenarios found in many other domains. This section starts with an overview of the prototype implementation, describing the various ESB configurations done to ensure uniformity and fairness of the results obtained. Thereafter, an experimental setup and comparative performance analysis of the ESBs is also presented.

4.4.1 Implementation Overview

The aim of the implementation was to investigate performance of Mule, ServiceMix and JBoss ESB with respect to service orchestration and dynamic service discovery. For service orchestration, ESB accepts the user-request on demand and triggers the business process which orchestrates different services to provide response to user-request. While for dynamic service discovery, the ESB dynamically discover all the services in the registry.

The prototype implementation presented was carried out using Java related frameworks and technologies. Java 2 Standard Edition running on Eclipse Helios was primarily used as the programming environment. Axis2 version 1.1 was used to implement all the web services orchestrated. Sequence diagram for LoanBroker scenario is shown in Figure 4.1. The scenario has six web services that are orchestrated to form the LoanBroker process. Two types of services were modeled and they are simple and composite or complex services. Simple services model service that cannot or do not need to be further decomposed because they are usually provided by external partners therefore no implementation is available for such services. Such services in our implementation are CreditAgency, Lender, Bank1, Bank2 and Bank3. For our Loan Broker we deployed simple services on Apache Tomcat server to mimic external partners. On the other hand a composite service is used to model business process and service which consume other services. These are usually represented as a workflow consisting of an ordered sequence of steps where each service is a call to another service. By default these series of steps are executed sequentially one after the other. In the implementation presented here, there is only one composite service, namely LoanBroker. LoanBroker composite service was deployed inside an ESB server. Finally, WSDL2Java tool was used to generate proxy stub used by ESB to invoke the service implementation of our process.

BPEL 2.0 standard was used to define the Loan Broker process. Apache ODE was used to provide runtime orchestration engine for the Loan Broker process. WSDL2Java tool was used to generate proxy stub used by ESB to invoke the service implementation of the LoanBroker process. Each of the three ESBs (Mule, ServiceMix and JBoss ESB) was configured to accept user requests as SOAP messages over HTTP.

4.4.2 Walk-through of the Loan Broker Application

LoanBroker application models a real world scenario of a consumer looking for the best loan quote by consulting a number of banks as shown in Fig.1. Message flow is as follows:

1) Consumer sends loan request with loan amount to LoanBroker.
2) LoanBroker sends request to Credit Agency to get credit profile.
3) LoanBroker receives credit profile.
4) LoanBroker sends request to Lender Gateway to determine the most appropriate lenders (banks) to contact based on consumer's credit profile and amount requested.
5) List of potential lenders is returned.
6) LoanBroker sends loan quote request to the potential lenders.
7) Each potential lender computes and returns interest rate.
8) Amongst the rates computed by potential lenders, the best rate is selected.
9) Best loan quote is returned to the consumer.

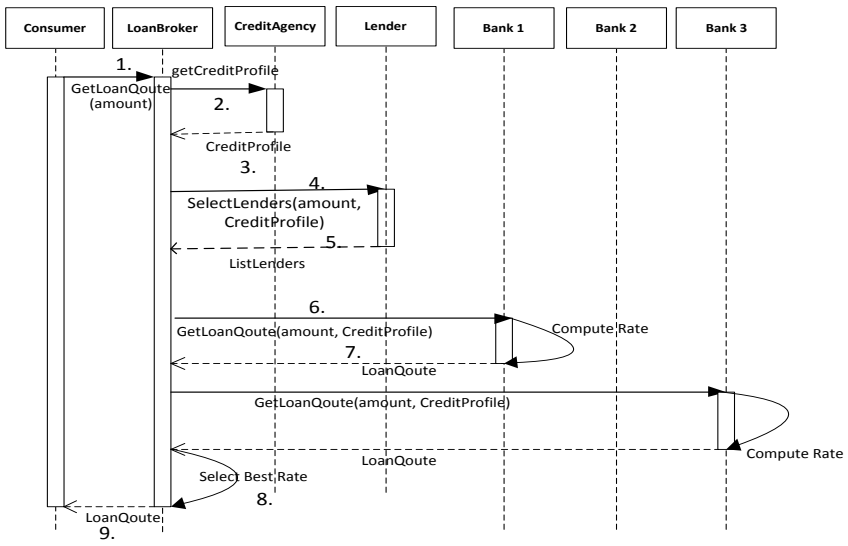

Figure 4.1: LoanBroker Sequence diagram

4.5. Integration Model Design for our ESB Performance Evaluation

In order to achieve a complete SOA, service composition and discovery mechanisms are required in addition to the ESB. This work aims to evaluate the performance of integrating ESB with UDDI and BPEL engine. Specifically, the following three key designs elements were evaluated:

- Simple ESB – Direct Service orchestration
- ESB integrated with BPEL engine for service orchestration
- ESB integrated with UDDI for dynamic service discovery

We now discuss each in turn.

4.5.1 Simple ESB – Direct Service Orchestration

Simple ESB is the basic design where ESBs perform service orchestration without any external component. In this design the composite service was hosted inside the ESB using Apache CXF and it was responsible for coordinating and directly invoking external services for the LoanBroker application. The purpose of this design was to investigate performance of each ESB when considering direct service orchestration. External services were deployed on the tomcat

server and they included CreditAgency, Lender, Bank1, Bank2 and Bank 3 service. Section 4.7.1 provides detailed configurations of each ESB.

4.5.1.1 Detailed configuration of each ESB

As it was previously mentioned that different ESB products uses different approaches towards achieving a certain integration capability. This led to different configurations as presented in this section.

4.5.1.1.1ServiceMix ESB

ServiceMix is based on JBI specification and it uses the concept of Binding Components (BC) and Service Engines (SE). The HTTP BC was configured to expose the CXF web service hosted inside ServiceMix. SOAP messages are exchanged between internal and external web services. Web service client send loan request and receive loan quote via HTTP protocol as shown in Figure 4.2 below.

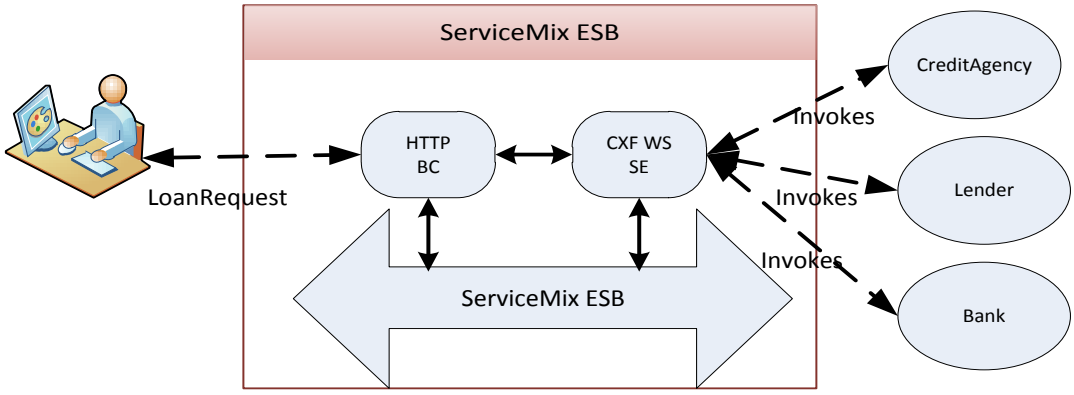

Figure 4.2 ServiceMix configurations for Direct Service Orchestration

4.5.1.1.2Mule ESB

Unlike ServiceMix which is based on JBI, Mule ESB uses Inbound and Outbound concepts. We have configured HTTP inbound endpoint to expose CXF web service hosted in Mule over HTTP, essentially making it an HTTP server. To invoke external web services SOAP outbound

was used to exchange SOAP messages as shown in Figure 4.3. CXF WS was used to host the LoanBroker composite service inside Mule ESB.

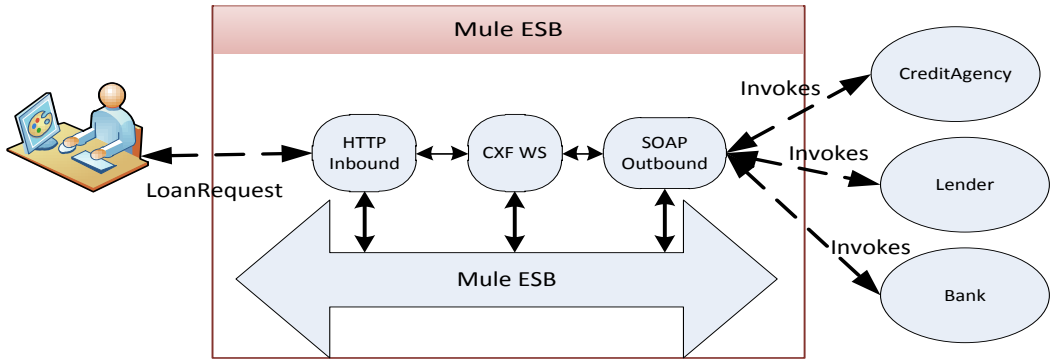

Figure 4.3 Mule configurations for Direct Service Orchestration

4.5.1.1.3JBoss ESB

JBoss ESB uses the concept of ESB unaware and ESB aware messages. Each service describes "Listeners" which listen to external messages and list "Actions" which are inside the ESB and are executed sequentially. The HTTP gateway was configured to listen to HTTP messages thereby exposing JBoss CXF web service to the clients. SOAP processor was used to process SOAP messages to and from internal web services. In addition, JBoss CXF WS invokes external CreditAgency, Lender and bank web services using SOAP messages.

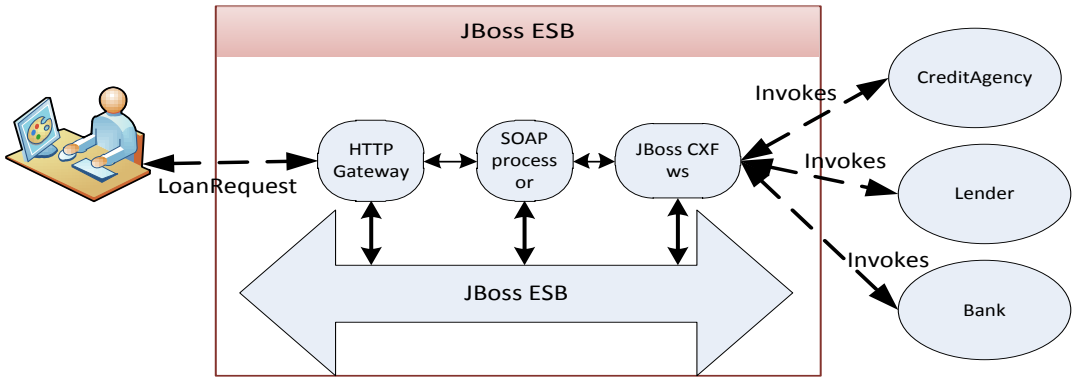

Figure 4.4 JBoss configuration for Direct Service Orchestration

4.5.2 ESB integrated with BPEL engine for service orchestration

Although not all ESBs provide native support for BPEL engines but there are APIs that exist to assist in integrating ESB and BPEL engine (Mulik*et al.*, 2009). In this design, Apache ODE was included as external BPEL Engine to all ESB in order to support execution of WS-BPEL processes. LoanBroker scenario was implemented as business process written using BPEL 2.0. The business process orchestrates and invokes CreditAgency, Lender and Banks web service. Integrating ESB and Apache ODE involved different configurations for each ESB. The configurations are detailed in the section below.

4.5.2 .1 Configuration of each ESB

This section presents different configurations of integrating ServiceMix, Mule and JBoss ESB with Apache ODE engine to provide support for BPEL business process execution.

4.5.2 .1 .1 ServiceMix ESB

CXF Service Engine was configured to register BPEL LoanBroker composite web serviceinto ServiceMix. Apache ODE has support for JBI, so for ServiceMix we had an option of deploying JBI distribution of Apache ODE inside the ESB. However for fairness throughout the experiment, it was kept external and integrated with ESB using Apache Axis2 distribution which was deployed on Apache tomcat. The configuration is depicted in Figure 4.5 below.

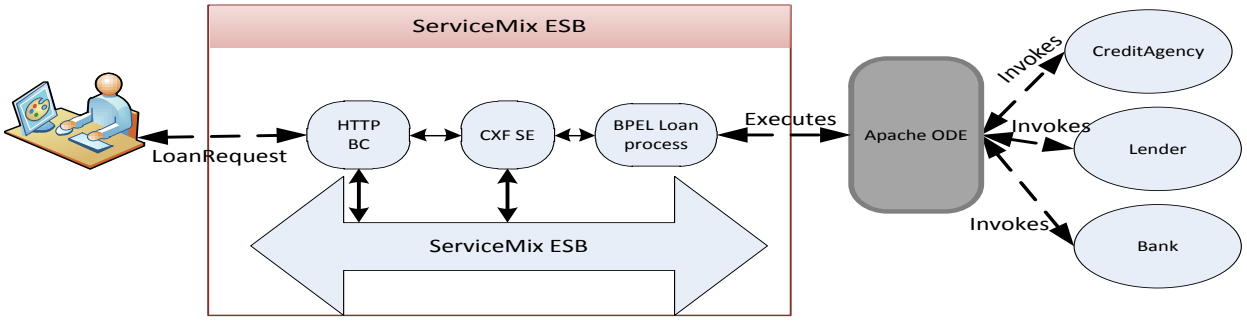

Fig 4.5 ServiceMix configuration for BPEL Orchestration

4.5.2 .1.2 Mule ESB

BPEL LoanBroker process was hosted inside Mule using CXF. To execute the process, Apache ODE was configured to communicate with the ESB using SOAP outbound component which sends messages from and to the engine. Same as the first scenario HTTP outbound was used to exposed web service. The configuration is depicted in Figure 4.6

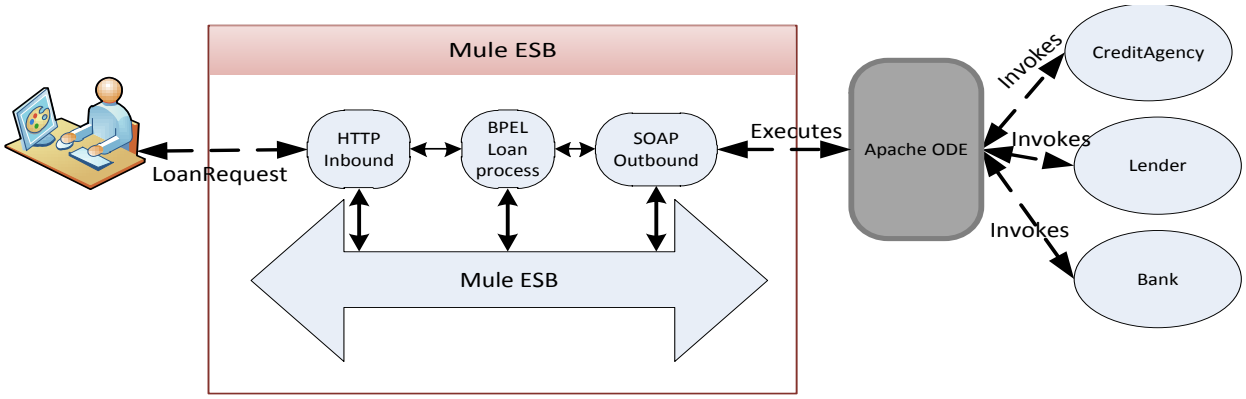

Fig 4.6 Mule configuration for BPEL Orchestration

4.5.2 .1.3JBoss ESB

JBoss ESB has native support for JBoss RiftSaw. However, Apache ODE was used for the experiments presented in this work as depicted in the Figure 4.7 below. BPEL Loan process was hosted inside the ESB and the runtime execution was provided using external Apache ODE. SOAP over HTTP was used for communication between the engine and the ESB hosting BPEL process.

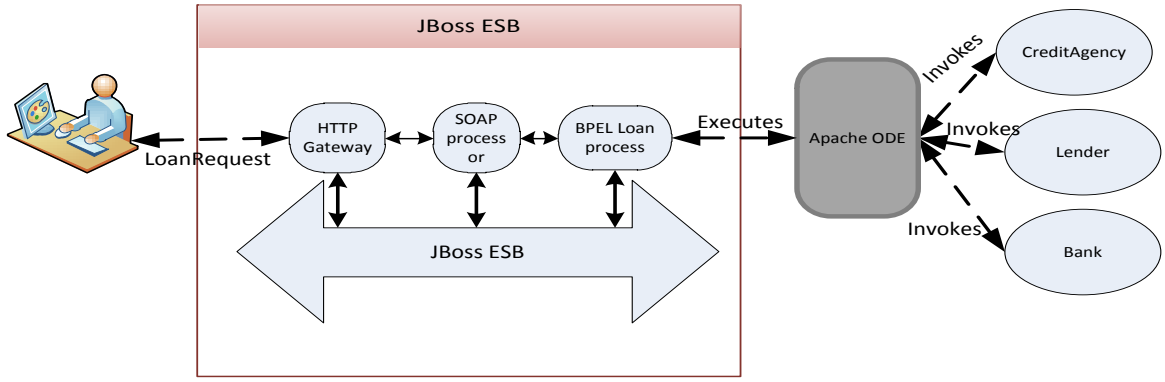

Fig 4.7 JBoss configuration for BPEL Orchestration

4.5.3 ESB integrated with UDDI for dynamic service discovery

In the dynamic environments like SOA, the number of services increases over-time. There is a need to discover the most appropriate service dynamically at runtime. The UDDI can provide dynamic service discovery capability in the ESB environment. This section presents an overview of the architecture for integrating ESB with UDDI registry. Then later give the detailed configuration of UDDI registry each ESB.

4.5.3.1High-level architecture for integrating ESB and UDDI server

The current ESB implementations support static discovery of services which is inflexible to change (Yu and Yan, 2011). Defining service routing at design time can result in too many issues including invocation faults when the service required is unavailable. In addition as number of services increases it can results to maintenance nightmare. So enabling dynamic service discover and selection in the ESB can be a solution to some of the issues posed by static service discovery. Integrating ESB with UDDI server can provide dynamic service discovery and selection by allowing service information to be registered in a central repository. Once the service is requested by the service consumer, an ESB get its address dynamically from UDDI server. Since the basic information about all services running inside the ESB is stored in the UDDI server, the service management becomes easier. Figure 4.8 below introduce the high level architecture of integration of ESB with UDDI service. The architecture has three layers and they are Business layer, Integration layer and ESB layer

- Business Layer is at the top and it includes service providers and consumers. In addition it contains enterprise application systems that map business processes to business services

- Integration Layer is at the middle and it contains two major components for integrating ESB and UDDI, these components are proxy and UDDI server. The proxy provides access to UDDI server and it can be implemented by a web application. The UDDI server is responsible for managing basic information of all services that are available in the ESB.
- ESB Layer is at the bottom and contains the ESB. This layer provides the facility for routing messages form source to destination application. In addition ESB contains dynamic service discovery mechanism for service look-up at runtime. Service consumers request services form ESB and then ESB query the UDDI to find the basic information about the service then bind with that service.

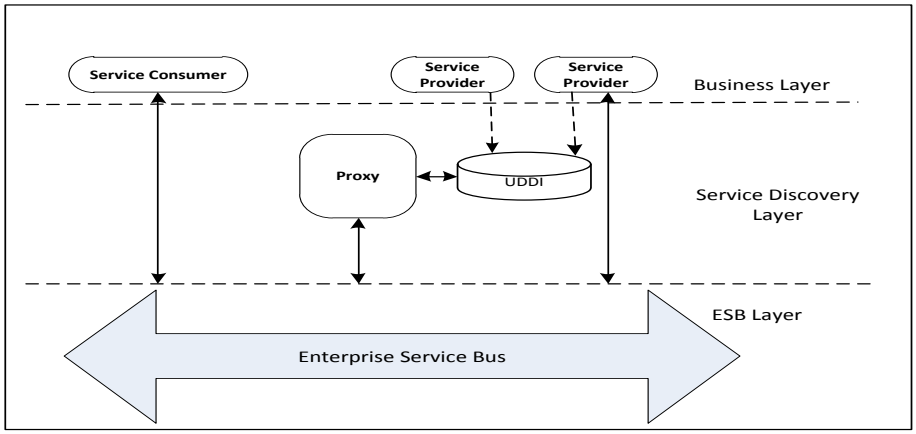

Figure 4.8: Overview of ESB integrated with UDDI

4.5.3.2Detailed configuration for each ESB

This section provides the configuration of ServiceMix, Mule and JBoss ESB integrated with jUDDI to enable dynamic service discovery.

4.5.3.2.1 Integrating ServiceMix ESB and jUDDI

ServiceMix ESB is based on JBI specification which defines Binding Component (BC) and Service Engine (SE). To expose LoanBroker service to external client, the HTTP BC was configured to accept SOAP request and relay response. Apache jUDDI client was used to provide access to the jUDDI registry while dynamic discovery mechanism that discover and sequentially invoke each of CreditAgencyStore, CreditAgencyBank, Lender and BankService is incorporated inside CXF SE. Then based on the service information discovered, CXF client is used to invoke the external service. BC and SE are packaged to form a single service assembly (SA) that is deployable to the ESB. Figure 4.9 shows how ServiceMix ESB was configured to enable service discovery.

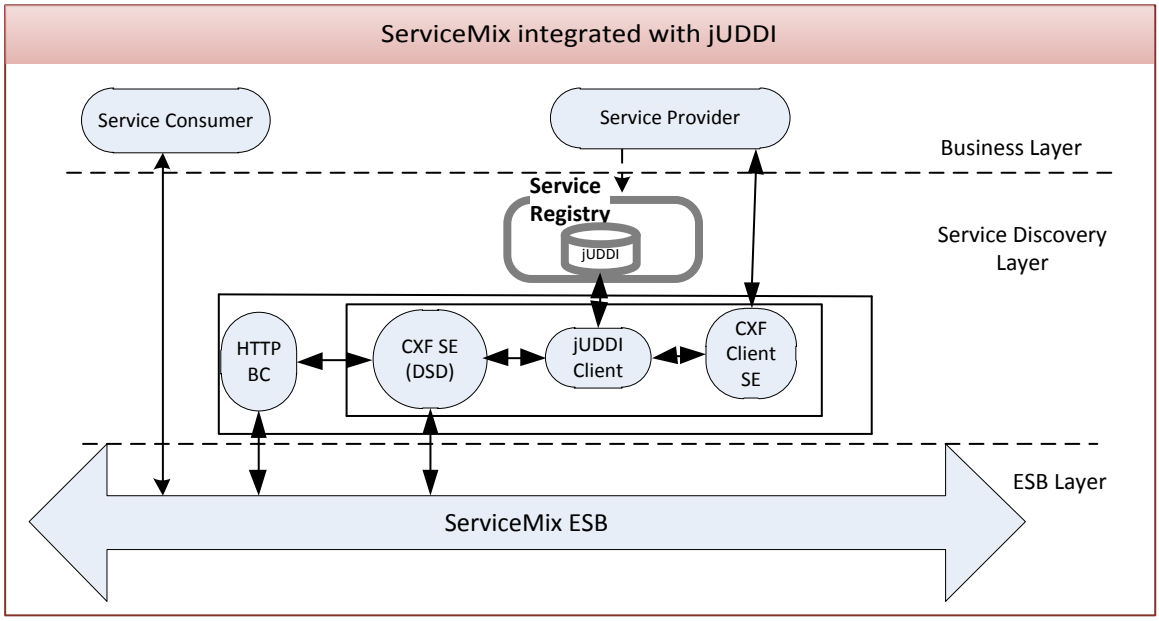

Figure 4.9: ServiceMix ESB integrated with UDDI

4.5.3.2.2 Integrating Mule ESB and jUDDI

Mule services consist of inbound router, service implementation component and outbound router. Inbound router exposes the service to the external entities and it's responsible for receiving and processing of messages sent by previous service or client. In this case

SOAP/HTTP inbound component was used as shown in the figure. The Component represents our LoanBroker application that invokes a CreditAgency, Lender and Bank services that are published in the jUDDI registry. LoanBroker component takes loan quote request from the user via SOAP/HTTP message then discover and invoke external services. Service implementation component provides proxy that link Mule with jUDDI using jUDDI client. In addition it contains the dynamic discovery mechanism that ensures services are selected according to their QoS information, specifically availability value. Then based on the information returned by jUDDI, the component uses CXF client to invoke the discovered web service. Finally the Outbound router receives the results and sends them to other services or applications. The Figure 4.10 shows the complete configuration of Mule integrated and jUDDI.

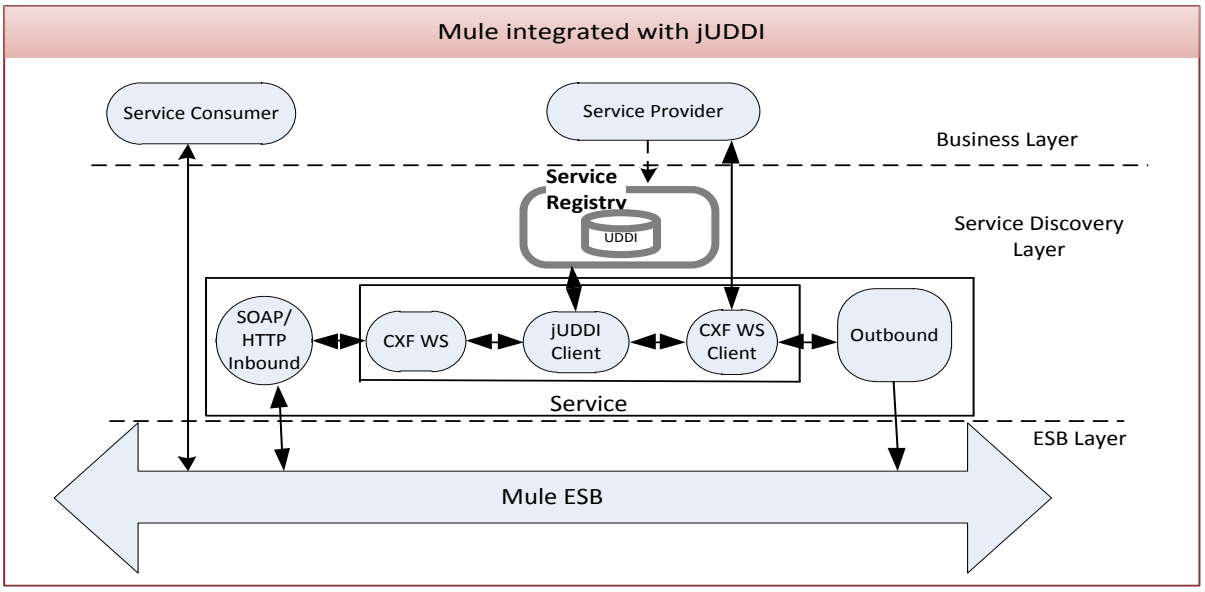

Figure 4.10: Mule ESB integrated withjUDDI

4.5.3.2.3 Integrating JBoss ESB and jUDDI

JBoss ESB has native support for Apache jUDDI, so we had an option of using an internal jUDDI implementation. However, for fair comparisons of the three ESBs, Apache jUDDI was kept external as shown in Figure 4.11. An ESB-unaware message was sent from service

consumer via HTTP gateway to SOAP processor which was configured to be the ESB-aware component. Upon receiving the message that contains loan quote request, the LoanBroker service discover about the available CreditAgencyStore, CreditAgencyBank, Lender, and Bank services. The discovery mechanism uses jUDDI client to interact with the registry. Invocation of these services takes place via JBoss CXF web service. Loan response is sent back to the requesting application or service.

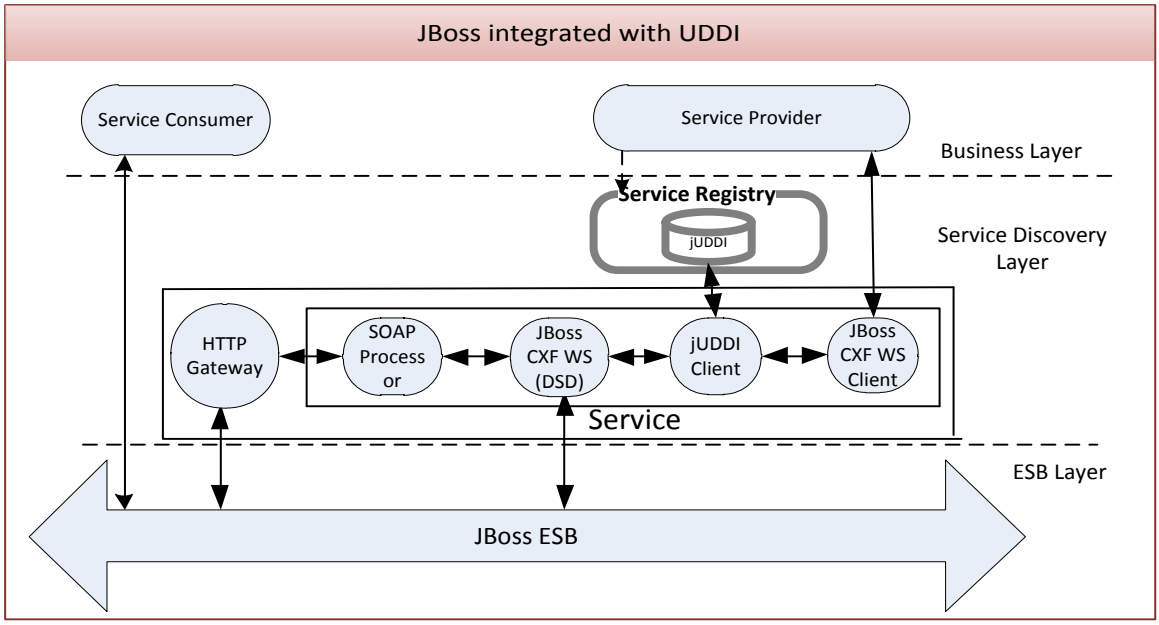

Figure 4.11: JBoss ESB integrated with UDDI

4.6. Performance Evaluation of ESBs

For the comparative performance analysis, we tested scalability, response time and throughput of each ESB using the configurations presented in section 4.5.

4.6.1 Scalability

Scalability is the ability of ESB system to handle increasing amount of work. Scalability is one of the important performance metrics in service-oriented systems because of the dynamic nature of the environment. The number of users requesting the same service can grow exponentially

from time to time which can impose performance overhead on ESB system. Scalability of each ESB was observed using response time.

4.6.2Average Response Time

Response Time is defined as the amount of time it takes for the system to return the desired response (Menascé, 2002). This time is usually measured by the client application sending request and receiving response. Then average response time was measured by taking average time of receiving response from a batch number of requests sent concurrently as seen in the formula. The aim was to measure how well the ESB system was able to respond to the increasing number of requests.

$Average\ Response\ Time = \frac{1}{n}\sum_{i}^{n}(Tf - T0)i$, where n is the number of requests

$$Tf: is\ the\ time\ when\ response\ was\ received$$

$$T0: is\ the\ time\ when\ the\ request\ was\ sent, and$$

$$i: represent\ request\ at\ index$$

4.6.3Throughput

The other most important performance measure of ESB system is throughput (Menascé, 2002). Throughput measures the number of transactions processed at a given time. This analysis measures throughput as the average transactions processed per second. The formula is as given below;

$$Average\ Transaction\ Time = \frac{1}{n}\sum_{i}^{n} TT$$

4.6.4 Statistical Analysis Method

After performing the experiments and collecting data to compare the performances of each ESB, we needed a method to analyze the results. Simply calculating the average response time or

throughput and generating graphs is not sufficient for the analysis that can lead to conclusions and decisions of which ESB performs better on which scenario. Therefore, Univariate ANOVA was used.

4.6.4.1 Univariate ANOVA

For our data analysis, we used Univariate ANOVA to test our null hypothesis that says all three ESBs would have a similar performance. Therefore, the mean difference between two ESBs compared over a set of matched pairs of data points would be zero (Tabachnick and Fidell, 2007). The P-Value threshold chosen for statistical significance was 0.05. Thus, if the calculated P-Value was below 0.05, then we reject null hypothesis, otherwise there is not enough information to reject null hypothesis. The hypothesis is as defined below:

$H_{0:}$ *All the three ESBs would have a similar performance*

$(\mu 1 = \mu 2 = \mu 3)$

$H_{1:}$ *At least one ESB's performance is different from the rest*

$(\mu 1 \neq \mu 2 \neq \mu 3)$

$(\mu 1 \neq \mu 2 = \mu 3)$

$(\mu 1 = \mu 2 \neq \mu 3)$

For each experiment below, a P-Value was calculated for each metric. The Univariate ANOVA helped to analyze the results obtained by focusing on a P-Value below the threshold of 0.05 as our set level of significance. In addition, the analysis was used to prove whether or not the performance of each ESB was equal to its peer, for a particular test.

4.6.5 Experimental Setup and Results Analysis

This section presents the results obtained when carrying out different experiments that aimed to investigate the performance each ESB. This investigation was motivated by the fact that these ESBs use different approaches towards realizing any ESB capability including service orchestration. So selecting a single ESB for service orchestration is rather a difficult task. Performance investigation was used to determine the ESB that perform better on service orchestration and dynamic service discovery scenario.

This section describes in detail all the experiments and analyses carried out and discusses the results that were obtained. We carried out various simulation experiments that were directed to investigating the following:

1) Direct Service Orchestration time as the number of concurrent requests increase.
2) BPEL service orchestration time as the number of concurrent requests increase.
3) Dynamic Service Discovery time as the number of service discovered and published increase.

4.6.5.1 Experiments for Direct and BPEL Service Orchestration

This section presents the experimental setup, results analysis and discussion for Direct and BPEL Service Orchestration.

4.6.5.1.1Experimental Setup for Direct and BPEL Service Orchestration

To mimic distributed environment, two tired architecture was configured with two machines. The machines were both Intel (R) Core (TM) processors with an i5 CPU @ 3.20 GHz and 3.00 GB of RAM. These machines were running Windows 7 with 32-bit Operating System. The first machine was used to represent the server running the backend services of LoanBroker

application. More specifically this machine was running ESB which in turn contains LoanBroker composite service and Apache ODE for providing runtime environment of BPEL processes. In addition Apache Tomcat server was used as container for simple services. The second machine contained front-end application that invokes LoanBroker by sending SOAP request and measure response time as well as throughput. Apache JMeter version 2.5.1 was used as client and load generator. For testing purposes in the context of service orchestration, the ESB configurations introduced in Section 4.5.1 and Section 4.5.2 was used. The two service orchestration methods were considered in order to examine the performance of each ESB. The first method is direct service orchestration which involves a composite service directly invoking the external simple services. In this method, the composite service was deployed inside each ESB while external services were hosted in an external application server to mimic distributed environment while allowing the ESBs to have access to similar services for fair comparison.

The second service orchestration method includes the use of BPEL defined processes which are executed by BPEL engine. The BPEL process was deployed in each ESB and executed by Apache ODE engine. For ServiceMix there was an option of deploying JBI distribution of Apache ODE inside the ESB but Axis2 distribution deployed on tomcat was used since all the three ESBs had support for this distribution. During initial testing it became clear that the three ESBs would need modification to their default pooling size before they would actually be able to run the test cases. For example ServiceMix default maximum pool size was set to allow only 32 concurrent HTTP requests. The number of requests was varied from 50 to 400 requests with the range of 50 and service orchestration time and throughput were metrics measured.

4.6.5.1.2Experiment I: Direct Service Orchestration

This experiment is used as a base case, where an ESB directly perform service orchestration without the help of an external component. Figure 4.12 shows response time for each ESB given a number of concurrent requests. We observe that as the number of requests increases all three ESBs maintained constant behavior regarding response time. This behavior of the response time in all ESBs seems to suggest that the ESB did not rich saturation and hence requests where being processed at a faster rate than they were being generated. However ServiceMix and Mule achieved almost the same and lower response time. Throughput is shown on Figure 4.13, where all ESBs started with low throughput but progressed steadily as the number of requests increases. Since the response rates in Figure 4.12 seemed to suggest that the ESBs did not reached saturation, the throughputs for 50 to 250 requests seem to be a function of the rate at which the request were being generated. As a result, throughput increased linearly with increases in the as the number of requests. A somewhat constant throughput was observed from 250 to 400 requests. This may be due to the fact that the ESBs were nearing saturation. As a result the throughput began to be a function of the ESBs and hence the expected constant throughput for each ESB.

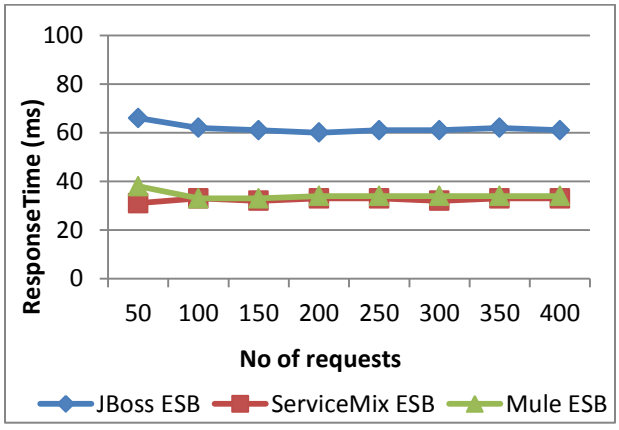

Figure 4.12: Response Time vs. No of Requests (Direct Service Orchestration)

Figure 4.13: Throughput vs. No of Requests (Direct Service Orchestration)

4.6.5.1.3Experiment II: BPEL Service Orchestration

The main aim of this experiment was to investigate the performance of each ESB when integrated with BPEL engine to support business process execution. Unlike in the first experiment, response time considering BPEL scenario increased as the number on requests increases. Therefore, integrating BPEL engine with ESBs had an effect on response time. This behavior shows how each ESB interact with Apache ODE. Regarding scalability as the number of requests increase, we observe that the rate at which the gradient change keeps widening as depicted in Figure 4.14.Therefore ServiceMix scale better than other ESBs while also achieving lowest response time. Considering throughput, we noted almost similar behavior with the first experiment. This means all ESBs were almost consistent with the processing of the transactions. However, ServiceMix had a better throughput in this scenario.

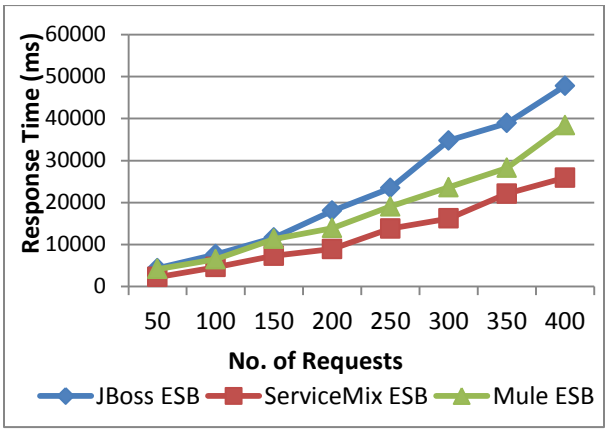

Figure 4.14: Response time vs. No of Requests (BPEL Service Orchestration)

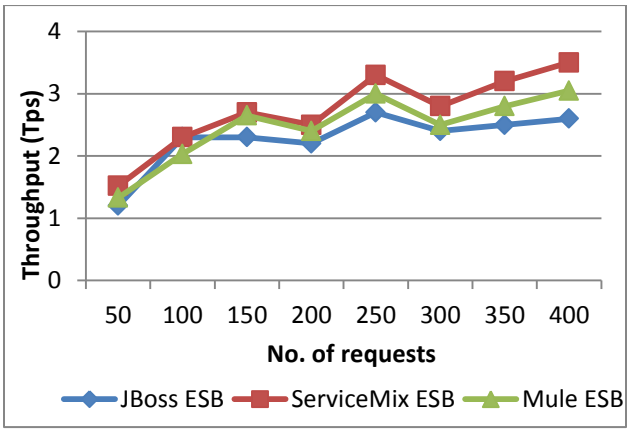

Figure 4.15: Throughput vs. No of Requests (BPEL Service Orchestration)

4.6.5.1.4 Direct and BPEL Service Orchestration Results discussions

Although the graphs above give us the visual representation of how each ESB performed in a given metric, a statistical analysis was needed to give meaning to the data collected. As stated previously in section 4.10.3, we ran Univariate ANOVA test on each metric and observed the P-Values that were below set threshold of 0.05. We tested hypothesis that were defined in Section 4.6.4

Table 4.1 shows the P-Values that were obtained from the post-hoc analysis of the Univariate ANOVA. Looking at the P-Values for direct service orchestration, throughput was significantly different for all the three ESBs, while response time for ServiceMix and Mule was comparable. Considering BPEL service orchestration scenario, the response time was significantly different for all ESBs, while on throughput ServiceMix had a throughput significantly different from that of Mule and JBoss. Thus looking at the computed values for our metrics in Table 4.2, ServiceMix obtained the lowest values for response time which are 32.5 *ms* and 12647.5 *ms* in both scenarios, while obtaining the highest throughputs which are 4.0125 *Tps* and 2.7275 *Tps* for direct and BPEL service orchestration respectively. Therefore, we can conclude that ServiceMix handled service orchestration better than Mule and JBoss ESB. The reason for ServiceMix better performance is that it has strong support for Apache ODE as an orchestration engine.

Table 4.1: P-Values for Direct and BPEL Service orchestration

ESBs	Direct Service Orchestration		BPEL Service Orchestration	
	Response Time	***Throughput***	***Response Time***	***Throughput***
JBoss vs. ServiceMix	0.000	0.007	0.000	0.000
JBoss vs. Mule	0.000	0.001	0.046	**0.060**
ServiceMix vs. Mule	**0.06**	0.000	0.033	0.013

Table 4.2: Means and grouping of ESBs for Direct and BPEL Service Orchestration

Means and grouping for Response time of Direct service orchestration	**Means and grouping for Throughput of Direct service orchestration**

ESB	N	Subset	
		1	2
ServiceMix	8	32.50	
Mule	8	34.25	
JBoss	8		61.75
Sig.		.060	1.000

ESB	N	Subset		
		1	2	3
Mule	8	3.1750		
JBoss	8		3.6500	
ServiceMix	8			4.0125
Sig.		1.000	1.000	1.000

Means and grouping for Response time of BPEL service orchestration

Means and grouping for Throughput of BPEL service orchestration

ESB	N	Subset		
		1	2	3
ServiceMix	8	12647.50		
Mule	8		18156.88	
JBoss	8			23321.00
Sig.		1.000	1.000	1.000

ESB	N	Subset	
		1	2
JBoss	8	2.2750	
Mule	8	2.4700	
ServiceMix	8		2.7275
Sig.		.060	1.000

4.6.5.2Experiments for Dynamic Service Discovery

The aim of this experiment is to investigate the scalability of the ESB in terms of the time it takes for dynamic service discovery and selection. This experiment was in turn divided into three test cases:

- Increasing number of service discovered.
- Increasing number of services published.
- Increasing number of concurrent requests.

4.6.5.2.1 Experimental Setup for Dynamic Service Discovery

In this experiment the two tired architecture was configured with two machines. The machines were both Intel (R) Core (TM) processors with an i5 CPU @ 3.20 GHz and 3.00 GB of RAM. These machines were running Windows 7 with 32-bit Operating System. The first machine was

used as the server running ESB and jUDDI registry. The second machine was used as the client that sends SOAP messages and obtaining response time and throughput. To evaluate only service discovery, the composite service was deployed inside each ESB, this service was then responsible for discovering and invoking all the LoanBroker services from jUDDI. A maximum of 20 services were discovered on the jUDDI registry and only 4 selected for loan process based on their QoS attributes. JBoss ESB comes integrated with jUDDI registry to support service registration, discovery and management but we chose not to use the already configured jUDDI since it might had impact on the results. So jUDDI was deployed on tomcat server and integrated with each ESB as an external component to allow a fair performance comparison. The experiment varied number of services discovered, published and concurrent requests to investigate performance and scalability of each ESB.

4.6.5.2.2Test Case I: Increasing number of services discovered

This test case investigates scalability of ESB as the increase in number of services that are discovered. The LoanBroker application consists of four services that are discovered in the jUDDI registry. At level-0 no services were discovered from the registry; Level-1 discovered only one service from the registry and so on until all the four services were discovered. Figure 4.16 shows the results obtained and it can be observed that as the number of service discovered increase, the response time also increases. All the three ESBs were equally scalable as the number of services discovered increases. The throughput is represented in Figure 4.17 and it shows that as the number services discovered increases the throughput of all ESBs decreases. Although this decrease in throughout exist JBoss ESB seemed to have better throughput than ServiceMix and Mule ESB.

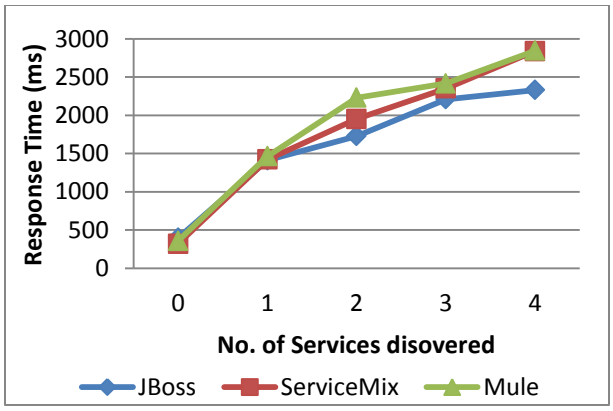

Figure 4.16: Response Time vs. No of services discovered

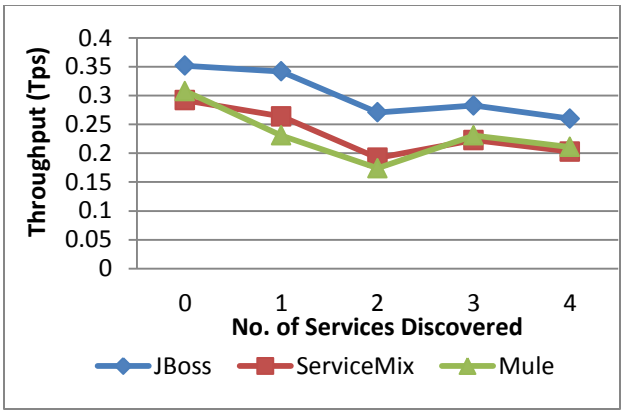

Figure 4.17: Throughput vs. No of services discovered

4.6.5.2.3Test Case II: Increasing number of services published

This test case presents the performance investigation of ESBs as the number of services that are published on the registry increases. In this case LoanBroker discover all its four services form the registry. At level-0 only four services were published in the registry, i.e. one per each service participating in the LoanBroker. At level-1 the number of services published was increased to eight, i.e. two identical services providing the same functionality but only one selected due to better QoS information. And so on until 20 services were published and five providing the same functionality. Figure 4-18 shows the response time obtained by each ESB when increasing number of services published. The response time was almost constant 12 to 20 services published. In view of this response time, the conclusion were made that all the ESBs were equally scalable because the change in gradient was almost the same. Figure 4.19 depicts the throughput obtained by each ESB in this test case. It can be observed that all the ESBs reached

their highest throughput when there were 8 services published in the registry

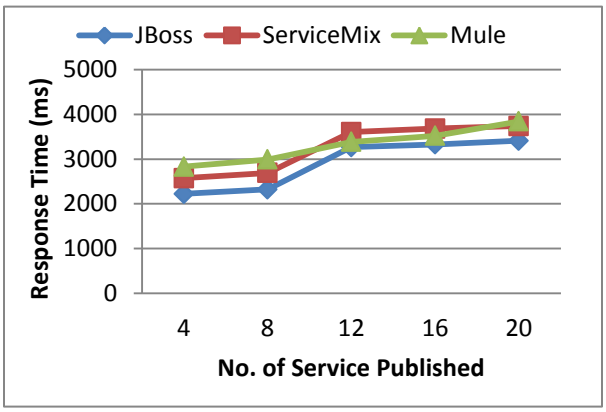

Figure 4.18: Response time vs. No of Services published

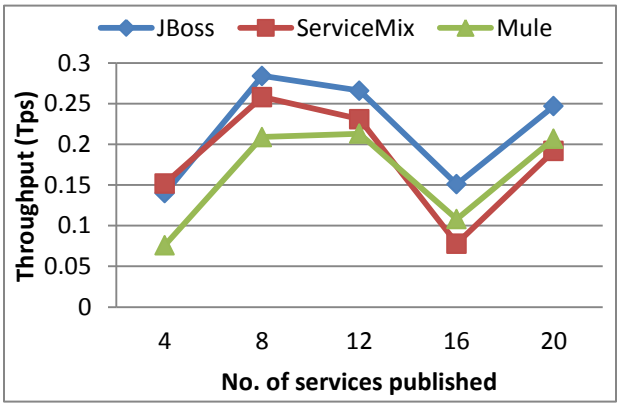

Figure 4.19: Throughput vs. No of services published

4.6.5.2.4Test Case III: Increasing number of concurrent requests

The aim in this experiment was to investigate the scalability of JBoss, ServiceMix and Mule ESB in terms of the time it takes to complete the service orchestration process as the number of concurrent requests increases. In this case, the number of services published in the UDDI held constant at 20. Among these services, only 4 services were selected based on their QoS value. This experiment investigates performance of each ESB under heavy load. The results obtained for response time are shown in Figure 4.20. As the number of requests increases, the response time for all the ESBs also increased. All the ESBs are also scalable when increasing number of requests. The throughput of each ESB is shown in Figure 4.21, it can be observed that as the number of requests increases the throughput also increase.

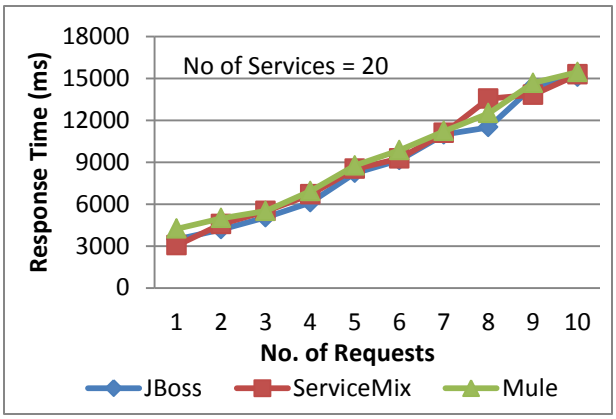

Figure 4.20: Response time vs. no of requests (Services published = 20)

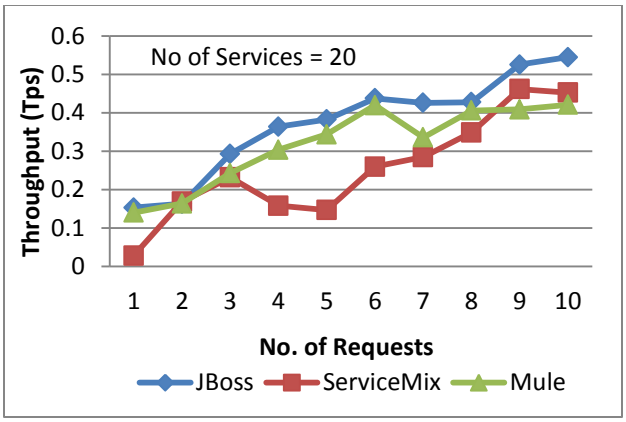

Figure 4.21: Throughput vs. no of requests (Services published = 20)

4.6.5.2.5 Dynamic Service Discovery Results Discussion

The graphs above present the findings for the performance of each ESB when considering dynamic service discovery using jUDDI as the registry. In addition to graphs, we performed a statistical analysis in order analyze data obtained and make conclusion on which ESB perform better in dynamic service discovery. Table 4.3 list the P-Values obtained for post hoc analysis for the three test cases of dynamic service discovery. Looking at the P-Values for increasing services discovered the response time was comparable for all ESBs, while we observe significant difference in throughput when comparing JBoss with ServiceMix and Mule. The computed means are shown in Table 4.4, JBoss had the lowest response time of 1616.20 ms and the highest throughput of 0.3016 Tps when increasing services discovered. The P-Values for increasing services published shows the difference in response time when comparing JBoss with ServiceMix and Mule. Throughput for ServiceMix was not significantly different than that of JBoss and Mule. Mean values shows that JBoss achieved the lowest response time of 2910.40 ms and the highest throughput of 0.2176 Tps. The P-Values for increasing number of concurrent requests shows that the response time for all ESBs was significantly differentwhile throughput for ServiceMix and Mule was comparable. Even in this test case JBoss obtained the lowest response time of 8497.88 ms with the highest throughput of 0.4093 Tps. Therefore, based on this

analysis it was concluded that JBoss handled dynamic service discovery better that the other ESBs. This is due to the fact that JBoss has a native support for UDDI. JBoss is shipped with Apache jUDDI for service registration, discovery and management (Conner *et al.*, 2012).

Table 4.3: P-Values for Dynamic Service Discovery

ESBs	**Increasing Services discovered**		**Increasing Services published**		**Increasing services published with Requests**	
	Response Time	Throughput	Response Time	Throughput	Response Time	Throughput
JBoss vs. ServiceMix	**0.247**	0.000	0.011	**0.093**	0.000	0.000
JBoss vs. Mule	**0.065**	0.000	0.05	0.013	0.000	0.000
ServiceMix vs. Mule	**0.635**	**0.925**	**0.806**	**0.410**	0.000	**0.431**

Table 4.4: Means and grouping of ESBs for Dynamic Service discovery

Means and grouping for Response time when **Increasing Services discovered**

ESB	N	Subset 1
JBoss	5	1616.20
ServiceMix	5	1777.40
Mule	5	1863.60
Sig.		.065

Means and grouping for Throughput when **Increasing Services discovered**

ESB	N	Subset 1	Subset 2
Mule	5	.231000	
ServiceMix	5	.234800	
JBoss	5		.301600
Sig.		.925	1.000

Means and grouping for Response time when **Increasing services published**

ESB	N	Subset	
		1	2
JBoss	5	2910.40	
ServiceMix	5		3258.20
Mule	5		3314.20
Sig.		1.000	.806

Means and grouping for Throughput when **Increasing services published**

ESB	N	Subset	
		1	2
Mule	5	.162600	
ServiceMix	5	.182200	.182200
JBoss	5		.217600
Sig.		.410	.093

Means and grouping for Response time when **Increasing number of requests**

ESB	N	Subset		
		1	2	3
JBoss	50	8497.88		
ServiceMix	50		9007.32	
Mule	50			9336.38
Sig.		1.000	1.000	1.000

Means and grouping for Response time when **Increasing number of requests**

ESB	N	Subset	
		1	2
ServiceMix	50	.318960	
Mule	50	.335220	
JBoss	50		.409300
Sig.		.431	1.000

4.7 Chapter Summary

This Chapter presented the comparative experimental results of JBoss, ServiceMix and Mule ESB with regard to service orchestration and dynamic service discovery. The goal was to investigate the performance of each ESB, and then based on the performance result select one ESB that handle each scenario better. Towards this goal, scalability, average response time and throughput was measured and used as the performance metrics. The results obtained were analyzed using statistical method and conclusions were made on each scenario. Overall, ServiceMix achieved better results on service orchestration. Therefore, scenarios that needs service orchestration can be better supported by ServiceMix. As an advantage ServiceMix is shipped with Apache ODE engine to support BPEL process execution inside the ESB. On the other hand, JBoss had the better ESB performance when considering dynamic service discovery using UDDI. In addition JBoss comes integrated with Apache jUDDI to support service registration, discovery and management. These two results were expected since the analysis in Chapter Three also advocated that ServiceMix and JBoss have strong support for service orchestration and dynamic service discovery, respectively.

To this end, we can conclude that there is no single best ESB solution for GUISET since different integration requirements are best supported by different ESB products. Therefore, to support all the GUISET integration requirements, a strategy to integrate different ESBs to work together was needed. ESB federation is a concept that emerged to conquer the limitations of a single centralized ESB model by allowing multiple physical ESBs to work together to form a single logical ESB. In the next chapter, we discuss this concept in details and introduce different patterns in which ESB federation can be achieved. The empirical evaluations of these patterns are also presented.

CHAPTER FIVE

ESB FEDERATION PATTERNS ANDPERFORMANCE EVALUATION

5.1Introduction

Despite the growth of using ESB technology to build new integration platforms, ESBs are still faced by challenges when employed for integration in large dynamic SOA environments. These challenges include the fact that large organizations are distributed in nature which makes it difficult to ensure single ESB throughout. In addition, the scalability of a single ESB is questionable when it has to support integration of a huge number of services and systems that constantly exchange messages for communication (Kumar*et al.,* 2011; Baude *et al.,* 2010; Gniel and Arnold, 2009; Nair, 2009). Chapter Three presented the analysis of the work that has been done related to the evaluation of ESBs towards supporting GUISET integration requirements. From this analysis, ServiceMix, Mule and JBoss ESB showed that they have some mechanisms that can be used to support GUISET integration needs. However, there is still a challenge of no single ESB that best support all the GUISET integration needs. For example, as presented in Chapter Three and Chapter Four, JBoss ESB performed well considering UDDI and dynamic service discovery support but comparably worse on the other requirements evaluated.

The challenge of different integration requirements that are best met by different ESBs calls for major adjustments in designing a GUISET integration platform. Industry practices are moving towards ESB Federation, so as to overcome the challenges imposed by the single centralized ESB in large environments. In an ESB Federation, two or more ESBs work together towards achieving integration needs of a certain organization. This work also employed ESB Federation approach to integrate ServiceMix, Mule and JBoss ESB in order todesign an environment that best support GUISET integration requirement. Federation of ESBs seemed to be a feasible

solution since it allowed different ESBs to work together, with each ESB used for the feature(s) that it best support.

There are three major patterns for ESB Federation that has been proposed (Dundek, 2010; Keen *et al.,* 2004). These patterns are Directly Connected, Hub-Spoke and Brokered ESB Federation patterns. These patterns are architecturally different but they all serve the same purpose of forming Federated ESB based on multiple ESBs working together to achieve integration requirements. The aim of this chapter was to use an empirical comparative analysis method to provide answers to the following questions:

1. Which ESB federation pattern has the best performance in terms of response time and throughput?
2. Does ESB federation have better performance compared to a single ESB in the GUISET context?

To successfully answer the above mentioned question, this chapter introduces the background and architecture of these patterns in Section 5.2. Section 5.5 and Section 5.6 discuss how these patterns were implemented to ensure that the GUISET integration requirements are best supported. More specifically, this was achieved by allowing each ESB participating in the federation to be used for the capability that it best supports. This study focused on only two GUISET integration requirements which are Service Orchestration and Dynamic Service Discovery. Section 5.7 presents the performance evaluation of three ESB federation patterns. ESB Federation pattern that obtained the best performance was then compared with a single ESB that was found to be closer in meeting GUISET integration requirements. This comparison is presented in Section 5.8. Finally, the chapter summary is presented in Section 5.9.

5.2Directly Connected ESB Federation Pattern

Directly connected ESB pattern is based on point-to-point topology. It allows ESBs to be connected directly to each other as peers. For example, the ESB that hosts the service consumer must know which ESB hosts the service provider, how to communicate with it in terms of what protocol to use and which format is expected to accept request. However, this defeats SOA principle of loose coupling. Some work has been done to federate multiple ESB domains using Directly Connected ESB Federation pattern. Dragicevic *et al.*,(2010) proposed an approach for cross-domain service integration called Declarative Inter-ESB Service-Connectivity Configuration Engine (DISCE). DISCE is a configuration engine that automates the connectivity required by applications accessing services on the distributed domains of SOA networked environment. DISCEmake services visible at the ESB domains by introducing a description realized as a WSDL file of the service in the registry of the client's domain.

Large SOA environments demand more scalable infrastructure to accommodate huge number of service consumers and providers. Enhancement of Petals distributed ESB toward an ESB federation is presented in Baude *et al.,* (2010). Petals distributed ESB contains ESB instances that are directly connected to each other as peers. The enhancement of Petals distributed ESB towards supporting large scale SOA environment include scaling service registries and message routers to the level of federation using hierarchical approach. Directly Connected ESB Federation pattern is known of its ability to scale well. However, with the routing information distributed among different ESBs, maintenance in this pattern would require exponentially more efforts as the number of links between ESBs increases (Keen *et al.,* 2004).

5.3 Brokered ESB Federation pattern

Rather than defining routing information within each distributed ESB as that of directly connected, a separate ESB called “broker” can be deployed. The function of the broker is routing information and acting as mediator between ESBs in the federation (Keen *et al.*, 2004). The broker ESB offers nothing other than routing services. This separates integration logic and related business rules from ESBs in the federation. For example, the ESB hosting service consumer need only to know how to communicate with the broker and the broker determines the ESB that host service provider. Therefore, each ESB has less implementation knowledge about other ESBs. This creates a loose coupled environment because the changes in one ESB will be less likely to affect the other ESBs.

Brokered ESB reduces number of point-to-point connection which eases maintenance nightmare that is a problem in Directly Connected ESB pattern. However, the major drawback of this pattern is that the central broker becomes an architectural bottleneck which introduces scalability, performance and fault tolerance problems (Callaway *et al.*, 2008).

5.4 Hub and Spoke ESB Federation Pattern

In this pattern, the ESB can either be Hub or Spoke. Spokes are connected to the Hub and they have no connection among themselves. The major difference between Hub and Spoke and Brokered ESB is that the Hub act as mediator and it can make requests or respond to requests as well unlike the Broker that does nothing other than routing. The Hub can be a fully-fledged ESB with enhance routing services to coordinate interaction amongst the Spokes in the federation. For example, a service consumer hosted in the Spoke sends a request to the Hub and the Hub check

whether it has service provider requested; if yes it respondto the request or else determine the Spoke that host required service provider (Dundek, 2010).

5.5 Design of Federated ESB

The implementation design is a three layered architecture representing five major components. These components are Service Consumer, Service Provider, Federated ESB, BPEL Engine and UDDI Registry. The five components work together to provide integration for the prototyped GUISET environments. The design is shown in Figure 5.1.

Design overview

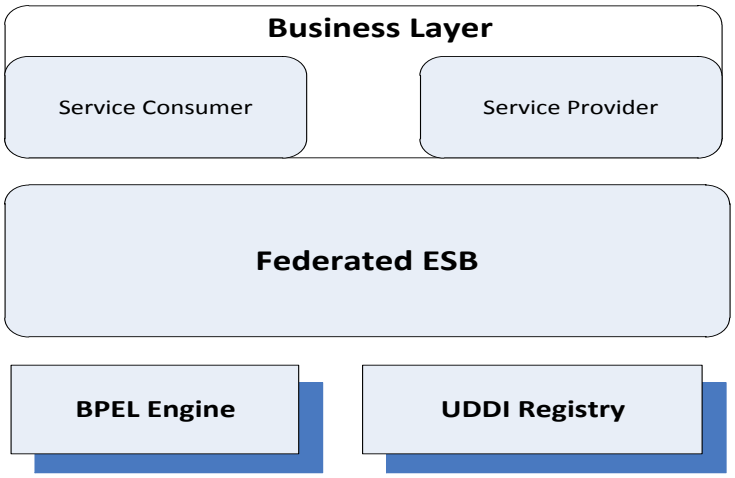

Figure 5.1: Overview Design solution for Federated ESB

Business Layer

This layer is responsible for providing access to the environment by means of allowing two entities to participate. These entities are service consumer and service provider.

Service Consumer – In the prototype environment service consumer is an entity that requires the capability offered by the service provider. The consumer's request is submitted through the

Federated ESB. A service consumer is assumed to have one or more tasks to be submitted for execution.

Service Provider – A service provider is an entity that develops and publishes web service that performs a given task. Web services are published together with their QoS information in order to differentiate between services offering the same functionalities. Each service published has QoS information. In this implementation QoS information was regarded as the "availability" value

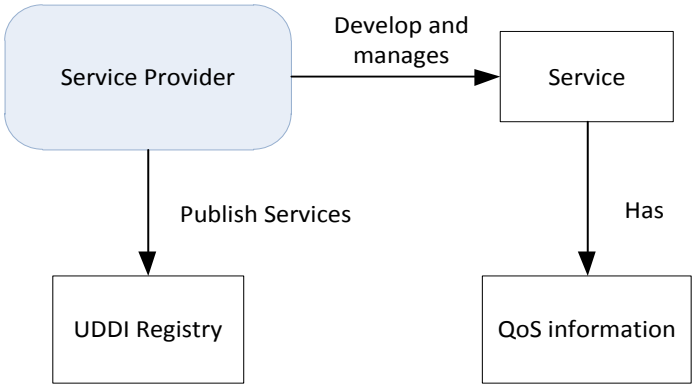

Figure 5.2: Service Provider Component

Federated ESB Component

Federated ESB is the main component responsible for coordinating the interaction of other components by:

- Accepting the consumer's request.
- Trigger BPEL process and submit the process to BPEL engine for execution.
- Dynamically look-up for service information by querying the UDDI registry.
- Based on the query results, the service is invoked.

Federated ESB component can be any of the ESB Federation patterns introduced in Section 5.2 to Section 5.4. These patterns differ in the way they route messages between ESBs. Figure 5.3 graphically represents the interaction of Federated ESB with other components. Once the request is made, Federated ESB initiates the BPEL process and dynamically gets the addresses of all the required services in the UDDI registry. Then using the service address, the service consumer binds and invokes the concrete service which is hosted in the Federated ESB.

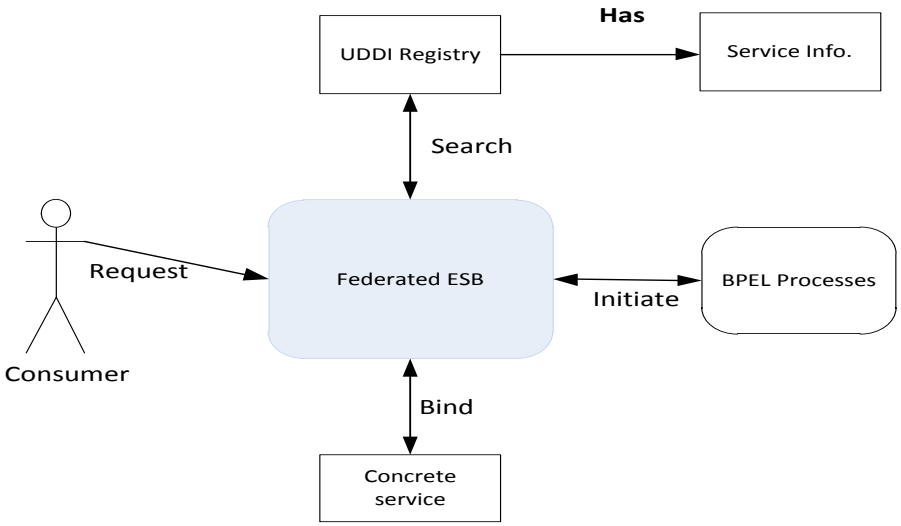

Figure 5.3: Interaction between Federated ESB and other Components

BPEL Engine Component

BPEL Engine component is responsible for providing runtime environment for BPEL processes. Figure 5.4 shows the BPEL engine and its related components. The BPEL written business process define the order of invocation for different services. These services are known using their WSDL interfaces. BPEL processes are submitted to BPEL engine for execution.

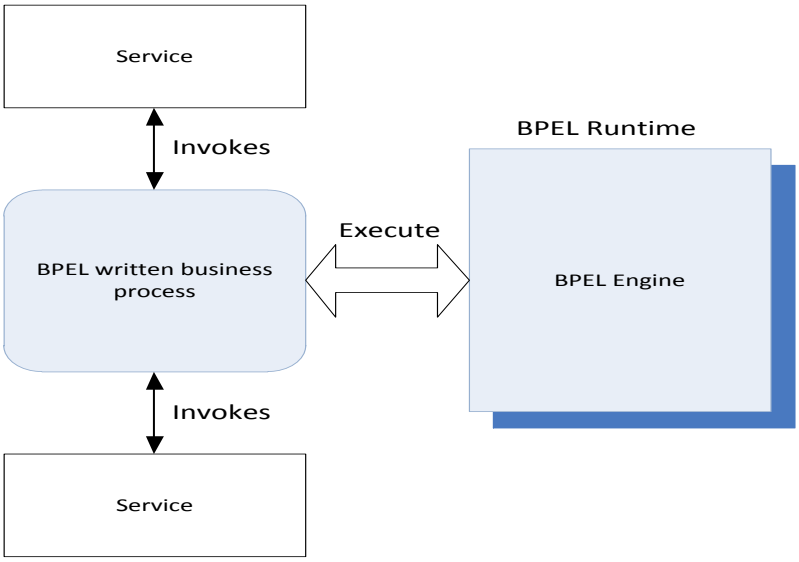

Figure 5.4: BPEL Engine Component

UDDI Registry Component

UDDI registry is the component responsible for storing information about the web services. This information is provided by service providers when they register their new web services in the registry so that they can be discovered by service consumers. Information stored in the registry includes business, service, and technical information that specify details on how to access the services offered. In addition, QoS information can be stored in the tModel data structure of the UDDI registry to assist in web service selection when more than one web services meet functional requirements requested. UDDI registry provides APIs for security policing, publishing, searching, updating web services. In an environment like Federated ESB some services might be hidden unintentionally from other ESBs. Therefore, UDDI can improve service visibility by providing centralized repository for publishing and searching for services. Moreover, since the information about the services is stored in the registry, service management becomes easier. Figure 5.5 shows UDDI registry component.

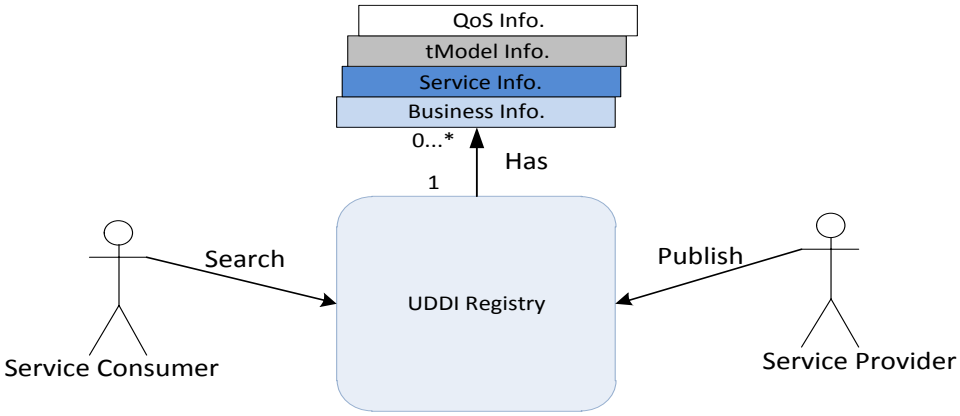

Figure 5.5: The UDDI Registry Component

5.6 Implementation of LoanBroker Scenario

Chapter Four presented the LoanBroker scenario which model the client obtaining the best loan quote by comparing loan rates from different banks. The scenario entails different services that are invoked to complete the loan quote process. These services are CreditAgencyBank, CreditAgencyStore, LenderService and BankService.

- CreditAgencyBank – takes customers Social Security Number (SSN) and compute banks credit history of the customer.
- CreditAgencyStore - takes customers Social Security Number (SSN) and compute Store credit history of the customer. The assumption is that the customer credit profile is obtained by adding banks and store credit history.
- LenderService – this service act as the gateway for the potential loan lenders. It is responsible for selecting Lenders (Banks) based on the customer's credit profile and the loan amount requested.

- BankService – This service is the gateway for Banks that are categorized as either High Rated or Low Rated. Each bank under the selected category computes and returns its name and loan rate.
    - High Rated – this category entails banks that offer large loan amount for customers with good credit profile.
    - Low Rated – This category has banks that offer small loan amount for customers with moderate credit profile.

5.6.1 Apache jUDDI Registry hosting LoanBroker Services

The above mentioned services were hosted by ESBs across federation, so as to allow ESBs to be active and contribute towards LoanBroker process. The information about these services was published in the jUDDI registry so that they can be dynamically discovered. The screenshot below (Figure 5.6) shows jUDDI portal containing LoanBroker services. Chapter Four presented the performance evaluation of Mule, ServiceMix and JBoss ESBs towards supporting dynamic service discovery. The results obtained favored JBoss ESB among other contenders. Moreover the aim of federating ESBs was to allow different ESBs to work together, with each being used for the feature that it best support. Therefore, in the Federation, JBoss was responsible for performing all service discovery needed by itself and on-behalf of other ESBs. JBoss was configuredto have direct communication with jUDDI Registry.

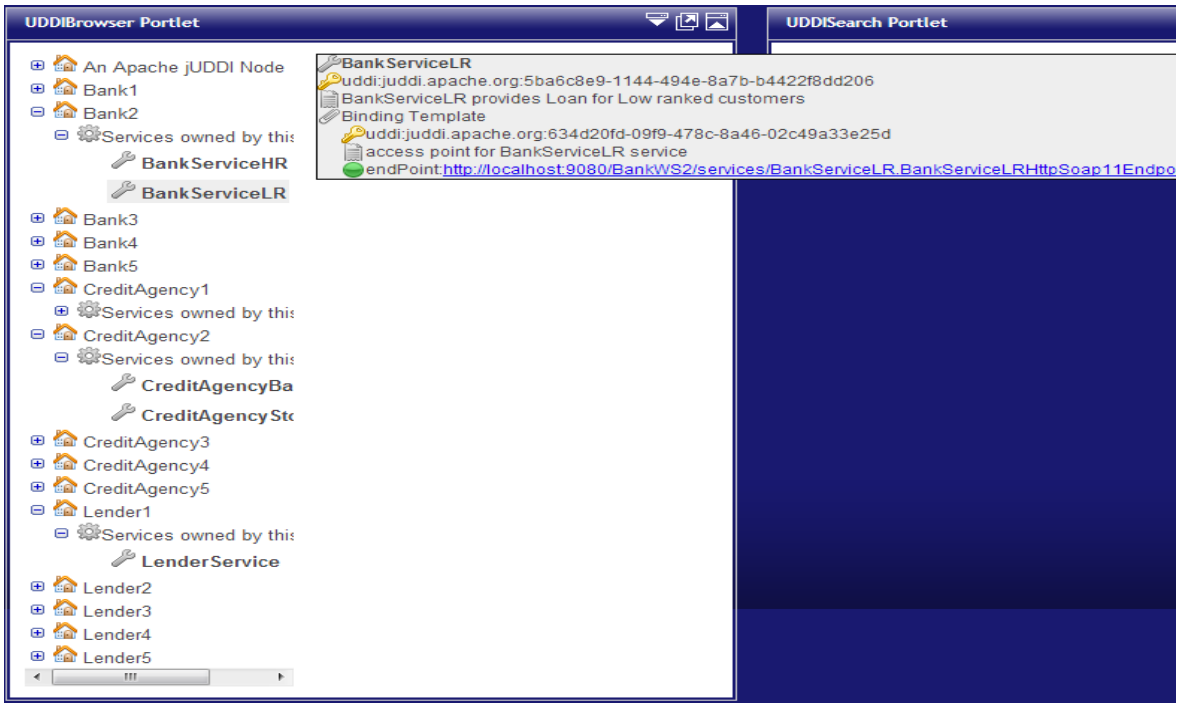

Figure 5.6: Snapshot of the Apache jUDDI showing LoanBroker services published

5.6.2 BPEL LoanBroker Processes

LoanBroker process involves the invocation of different services that communicate and exchange messages until the loan quote is obtained. WS-BPEL 2.0 was used as the design tool to define the LoanBroker process as shown inFigure 5.7 below. Chapter Four presented the performance evaluation of Mule, ServiceMix and JBoss ESB towards support of BPEL service orchestration. The results presented favored ServiceMix among other contenders. Therefore to achieve maximum results in the Federated ESB, ServiceMix was responsible for performing hosting and executing BPEL processes. ServiceMix was configured to have direct communication with Apache ODE to assist in BPEL process execution.

Figure 5.7: BPEL defined LoanBroker process

5.6.3 Implementation of Federated ESB Patterns

In the literature, three major ESB Federation patterns were proposed. At the beginning of this Chapter, an overview of Directly Connected, Hub and Spoke, and Brokered ESB Federation patterns was presented. This section presents the detailed implementation of each pattern. The LoanBroker business process was used for proof of concepts and performance evaluation of three federation patterns introduced. The aim was to recommend a single federation pattern for GUISET like environments.

5.6.3.1 Directly Connected ESB Federation Pattern

In this pattern, ESBs communicate directly with each other as shown in Figure 5.8. As mentioned previously that the LoanBroker services were distributed among different ESBs in the federation. More specifically, CreditAgencyBank was hosted inside ServiceMix, while CreditAgencyStore and LenderService were hosted by Mule, and JBoss hosted the BankService. In addition ServiceMix was configured to host the BPEL LoanBroker process that dynamically discover and invokes different services. Note that the LoanBroker process does not know CreditAgencyBank and CreditAgencyStore, it invokes CreditAgencyService. CreditAgencyService aggregates the results obtained from CreditAgencyBank and CreditAgencyStore to make a single customer's credit profile. CreditAgencyService is hosted by JBoss ESB and it enables the interaction between JBoss and the other two ESBs. Moreover, JBoss ESB host the dynamic service discovery mechanism.

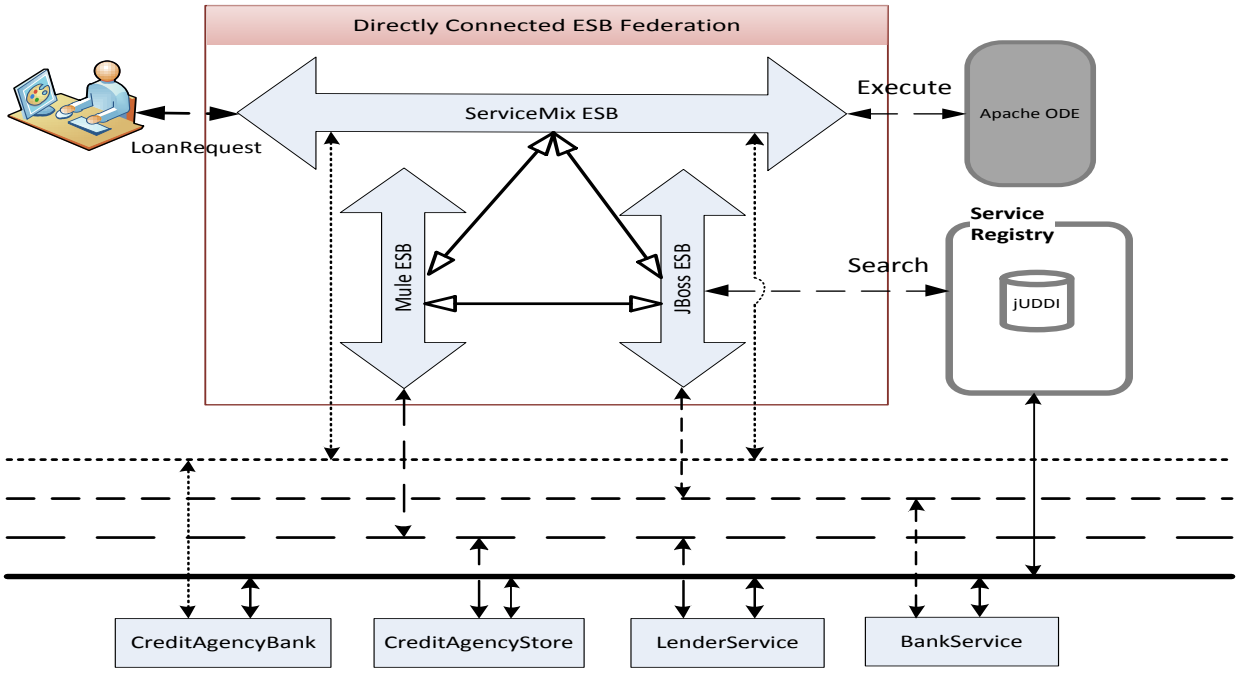

Figure 5.8: Directly Connected ESB Federation implements of LoanBroker

5.6.3.2 Hub and Spokes ESB Federation Pattern

This pattern is almost the same as Directly Connected ESB Federation. The major difference is that Spokes communicate via Hub. So when JBoss aggregates credit profile, CreditAgencyService sends request to CreditAgencyStore proxy hosted by ServiceMix. This proxy then invokes CreditAgencyStore on behalf of CreditAgencyService. Therefore, in Hub and Spokes communication between spokes takes place through Hub. Figure 5.9 shows the complete configuration of Hub and Spoke ESB federation pattern.

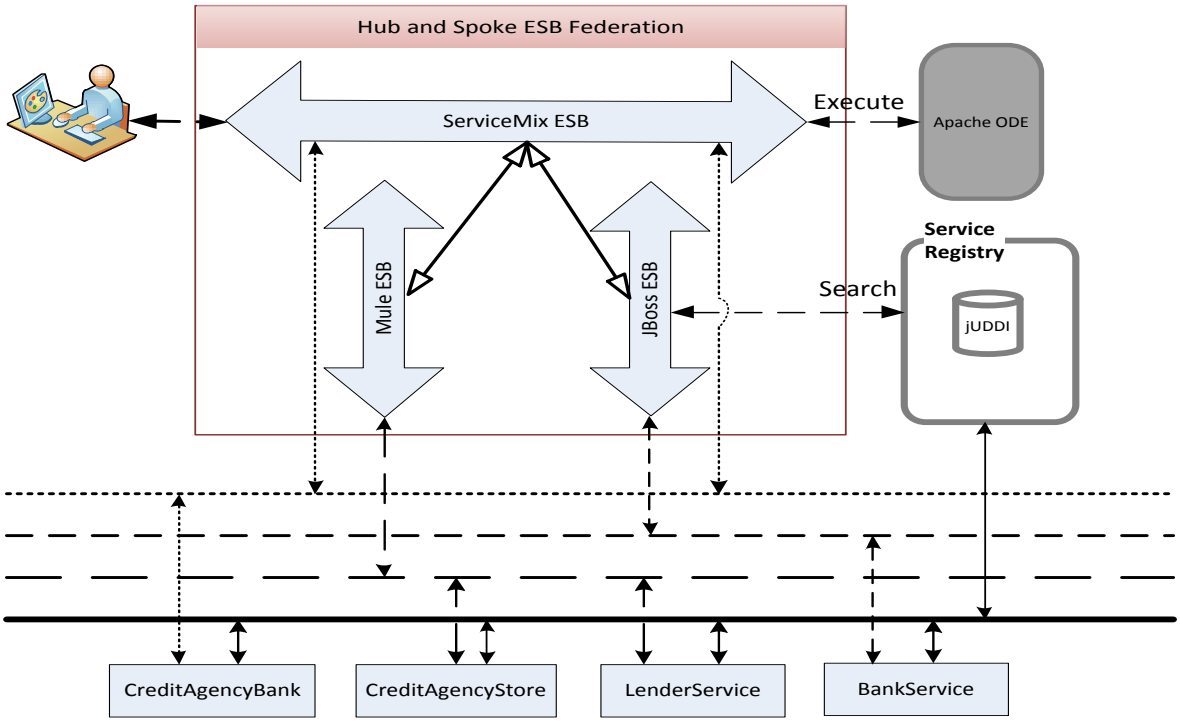

Figure 5.9: Hub-Spoke Federation pattern implements LoanBroker

5.6.3.3 Brokered ESB Federation Patterns

In this Federation pattern, there is a single ESB called broker which is dedicated for only communication. Therefore, ServiceMix was configured to be the broker ESB that host all the communication protocols for routing messages throughout the Federation as shown in

Figure5.10. In addition, ServiceMix hosts LoanBroker process. Moreover, CreditAgencyBank that was hosted by ServiceMix in the previous patterns was moved to JBoss ESB.

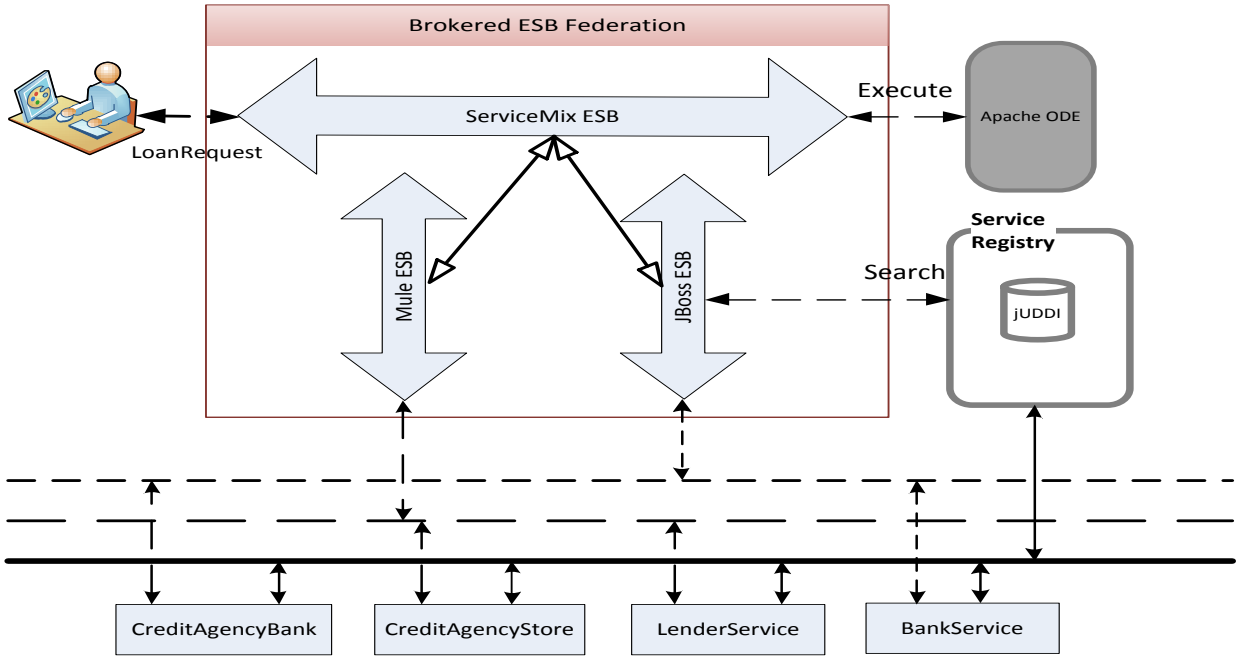

Figure 5.10: Brokered ESB Federation Pattern implements LoanBroker

5.7 Performance Evaluation of Federated ESB Patterns

This section presents the performance evaluation of Directly Connected, Hub and Spoke and Brokered ESB Federation patterns. The implementation of these patterns was provided in Section 5.6. However, it is noteworthy to mention that our implementation was restricted to just an experimental prototype with only manageable number of services deployed on the ESB and published on our local service registry. While, this implementation allows us to gain valuable insight of how GUISET integration requirements can be best supported, there was a need to further comparatively evaluate the performance of the three federation patterns in a controllable and repeatable environment, so as to ascertain their performance tradeoffs. The results of this evaluation will be used to recommend a single ESB Federation pattern for the GUISET

environment. Towards this end, this section presents an empirical evaluations and analysis of Federated ESB patterns.

The following key performance questions were investigated:

- What is the effect of increasing number of service discovered on the service orchestration time and throughput?
- How does increasing number of services published affect service orchestration time and throughput?
- What effect does increasing number of concurrent requests has on service orchestration time and throughput?

5.7.1Basic Assumption of the Simulation Model

In configuring our experiments, the following basic assumptions were made:

- We assumed that LoanBroker web services were already published by their service providers.
- Service Providers defined service QoS Attributes, specifically "availability" during service registration.
- Assumed the network delay was constant throughout the experiment.

5.7.2 Simulation Setup and Environment

To mimic distributed environment for our experiments, three different machines running 32- Bit Windows 7 Professional were used. The machines had Intel (R) Core (TM) *i*5 CPU with 3.20 GHz processor speed and 3 GB of RAM. The first machine was configured to run ServiceMix ESB which host LoanBroker BPEL process, Apache ODE for BPEL process execution, and Apache JMeter which was used as client that sends SOAP messages to LoanBroker process and measures response time and throughput. The second machine was running JBoss ESB which

hosts dynamic service discovery mechanism, and Apache jUDDI which was used as repository registry storing information about LoanBroker web services. Finally, the third machine was running Mule ESB hosting implementation of some of the LoanBroker web services. All the three machines were connected using 100 Mbps network switch. Upon initiation, the LoanBroker process consults JBoss ESB to dynamically discover services to be invoked. When there is a list of matching services then QoS value of availability was considered for service selection. The maximum number of services published in the jUDDI repository was 20, with each of the LoanBroker services having 5 replicas that have different QoS value of availability. A LoanBroker request was executed 10 times in each test case to investigate service orchestration time and throughput.

5.7.4 Experimental Results and Discussion

The implementation design of Directly Connected, Hub-Spoke and Brokered ESB federation patterns was discussed in Section 5.6. Based on this implementation, this section discusses the results obtained from various simulation experiments that were directed towards investigating the service orchestration time and throughput in the following situations:

- When the number of services discovered increases.
- When the number of services registered increases.
- When the number of concurrent requests increases.

The experiments were designed to test scalability, average response time and throughput of executing BPEL LoanBroker process. BPEL LoanBroker process dynamically discovers its services from jUDDI registry and invokes these services by binding with the service providers. In this case service providers host their services inside the ESBs.

The average response time is given as the sum of *n* service requests response time (*RT*) divided by *n* number of requests sent. *R T*is the time it takes to complete service orchestration process. This includes the service discovery and invocation time. The equation is as shown below;

$$Average\ Response\ Time = \frac{1}{n}\sum_{i}^{n}(RT)$$

Whereas the average transaction time is given by the sum of *n* service requests processed per second divided by the *n* requests. This is represented by the following equation;

$$Average\ Transaction\ Time = \frac{1}{n}\sum_{i}^{n}TT$$

5.7.4.1 Experiment I: Increasing number of services discovered

The aim of this experiment was to investigate the scalability, response time and throughput of the ESB federation patterns when the number of services discovered increases. In this experiment at Level- 0 all services were directly invoked by the LoanBroker process, therefore, no service discovery took place at this level. At Level – 1, only one service was discovered from the jUDDI registry, while the rest of services were directly invoked. The number of services discovered was increased until all the four services were dynamically discovered for the registry. The results obtained from this experiment are graphically represented in Figure 5.11 and Figure 5.12 and they are discussed in details in the results discussion section. The results in Figure 5.11 shows the response time of the three ESB Federation Patters (Directly connected, Hub and Spoke and Brokered ESB Federation). It can be observed that as the number of services discovered increases the service orchestration time also increase. Such behavior was expected, given the fact that searching the registry takes time due to two factors; first the registry needs to perform

authentication process on the request. The second factor being the fact that registry returns a list of matching service and the service discovery mechanism has to evaluate and compare services QoS information specifically "availability" as attribute was considered for service selection. The service with the high availability value gets selected to participate in the LoanBroker process. It can also be observed that all the three federation patterns were equally scalable in this scenario. Figure 5.12 depict the findings of our experiments for throughput versus number of service discovered. The results show that the throughput for all the ESB Federation patterns were nearly the same when there was no service discovery. The constant behavior from 0 to 3 services discovered was observed for Directly Connected and Brokered ESB Federation patterns

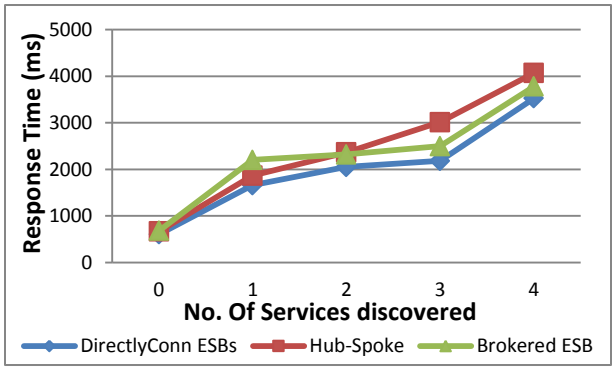

Figure 5.11: Response Time vs. No of service discovered for The three ESB Federation Patterns

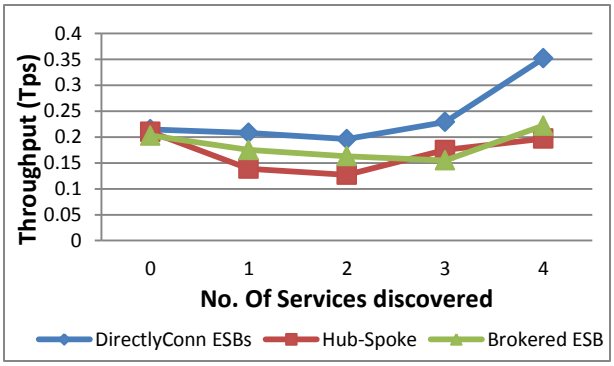

Figure 5.12: Throughput vs. No of service discovered for the three ESB Federation Patterns

5.7.4.2 Experiment II: Increasing number of service registered

This experiment aimed to investigate scalability of Federation pattern with increases in the number of services registered in the registry. LoanBroker process dynamically discovers and invokes four different services that exchange messages to complete loan process. At Level – 0, four services were registered in the registry and discovered by LoanBroker. At level – 1, we increased number of services in the registry by registering replicas of each service of the 4 services. Therefore, at Level – 1, number of services registered and discovered by the LoanBroker were increased to 8. Then, we increased the number of services registered in the

registry until the sum of 20 services were registered and discovered in the registry. The results obtained from this experiment are shown in Figure 5.13 and Figure 5.14. Results showing response time versus number of service registered is depicted in Figure 5.13. Figure 5.13 shows that increasing number of service registered had an effect on service orchestration time. The effect is in such a way that increasing number of service registered also increase number of response time. The lower response time at the beginning was caused by the fact that the registry was responding with only one service matching the requests as opposed to the list, therefore no comparison of QoS attributes needed. We observed from Figure 5.13 that the three federation patterns were equally scalable. Figure 5.14 shows throughput versus number of service published results. Throughput starts high and decrease then when reached 16 services registered it then increases again.

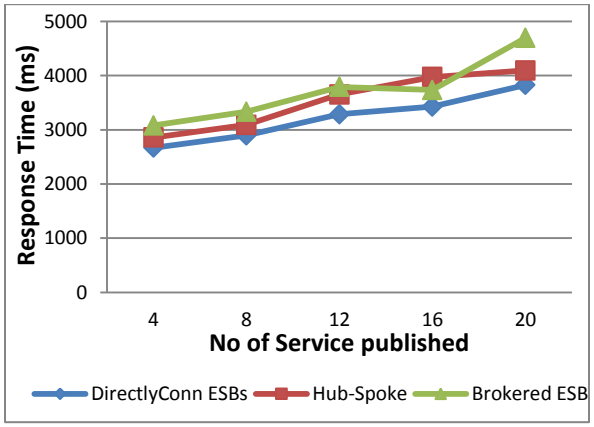

Figure 5.13: Response Time vs. No of service published for three Federated ESB patterns

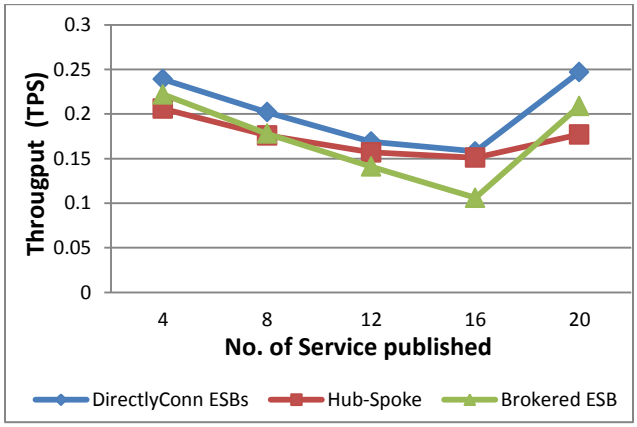

Figure 5.14: Throughput vs. No of service published for the three Federated ESB patterns

5.7.4.3 Experiment III: Increasing number of service registered

In SOA environments, it is always possible that two or more clients can be concurrently consuming the same service, so the environment should be able to behave the same regardless of the load. In this experiment, we investigated the behavior of each pattern when the number of concurrent requests increases. The results obtained are graphically represented in Figure 5.15and

Figure 5.16. The results in Figure 5.15 show that increasing number of requests was directly proportional to the response time. We also observed that all the three patterns were equally scalable as the number of requests increases. The throughput on the other hand was below 1 Tps in all test cases as shown in Figure 5.16. Having these results was not enough to make decision of which federation pattern performs better. Therefore, the results were further analyzed in Section 5.7.5.

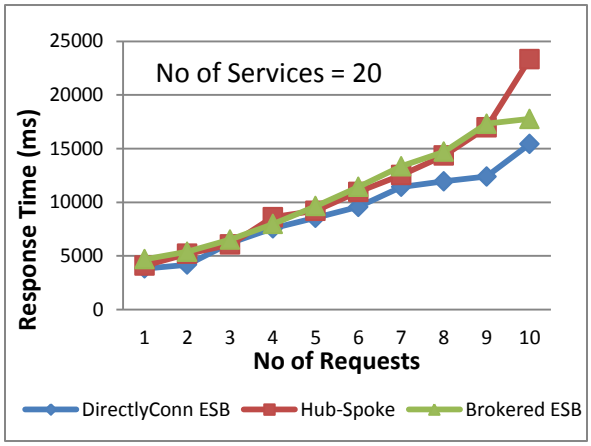

Figure 5.15: Response time vs. increasing no of requests (20 services published)

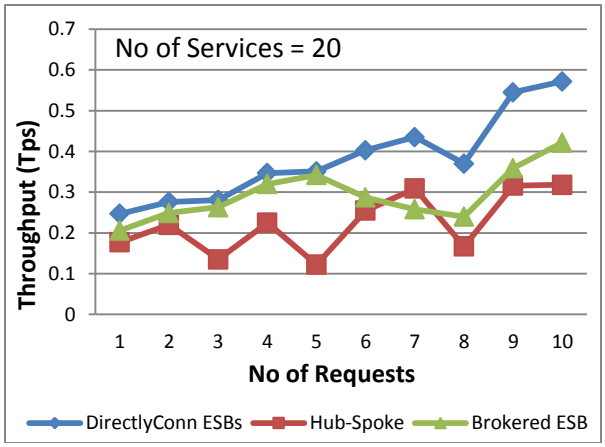

Figure 5.16: Throughput vs. increasing no of requests (20 services published)

5.7.5 Results Discussion

Graphical representation of the results obtained was given in section 5.7.4above. Although the graphs give visual representation of how each federation pattern performs in a given scenario, the statistical analysis was needed to assist in selecting one pattern that performed better. The analysis was based on looking at the P-Value at the 95% confidence interval, if the P-Value obtained was less than 0.05 then performance on the two candidates compared was significantly different. The P-Values for all experiments are recorded below in Table 5.1. We further recorded the means and the grouping for patterns in Table 5.2.

Looking at the increasing service discovered response time and throughput show that Directly Connected ESB pattern had significantly different performance than Hub-Spoke. Considering the experiment of increasing services published, we observed the difference in response time when comparing Directly Connected ESB with Hub-Spoke and Brokered ESB. Throughput for Directly Connected ESB was different with that of Brokered but comparable with that of Hub-Spoke. Looking at the scenario of increasing requests, the response time and throughput showed that Directly Connected ESB had significantly different performance compared to other federation patterns.

Table 5.2 represents means and clustering of the patterns. According to the computed values, Directly Connected ESB achieved the lowest response time of 2005.40 ms, 3221.60 ms, and 8452.92 ms in all three experiments respectively. Considering throughput, also Directly Connected ESB obtained the highest throughput of 0.2400 Tps, 0.2030 Tps and 0.4299 Tps in all the experiments respectively. The reason for such performance was due to the fact that in Directly Connected federation pattern, ESBs communicate directly with each other in a point-to-point model. This reduces time taken to perform message routing proxy as required by other patterns. Therefore, based on these analyses we can conclude that Directly Connected ESB federation pattern performed better in all scenarios followed by Hub and Spoke and then Brokered ESB.

Table 5.1: P-Values for comparison of ESB Federation patterns

Federated ESB Patterns	**Increasing Services discovered**		**Increasing Services published**		**Increasing services published with Requests**	
	Response Time	Throughput	Response Time	Throughput	Response Time	Throughput

DirectlyConn vs. Hub-Spoke	0.030	0.018	0.037	**0.055**	0.000	0.000
DirectlyConn vs. Brokered	**0.100**	0.049	0.003	0.041	0.000	0.000
Hub-Spoke vs. Brokered	**0.704**	**0.763**	**0.209**	**0.977**	**0.774**	**0.811**

Table 5.2: Means and grouping for ESB Federation patterns

Means and grouping for Response time when **Increasing Services discovered**

Patterns	N	Subset 1	Subset 2
Directly Conn	5	2005.40	
Brokered	5	2296.60	2296.60
HubSpoke	5		2396.40
Sig.		.100	.704

Means and grouping for Throughput when **Increasing Services discovered**

Patterns	N	Subset 1	Subset 2
HubSpoke	5	.169600	
Brokered	5	.183600	
Directly Conn	5		.240000
Sig.		.763	1.000

Means and grouping for Response time when **Increasing services published**

Patterns	N	Subset 1	Subset 2
Directly Conn	5	3221.60	
HubSpoke	5		3535.40
Brokered	5		3727.40
Sig.		1.000	.209

Means and grouping for Throughput when **Increasing services published**

Patterns	N	Subset 1	Subset 2
Brokered	5	.171200	
HubSpoke	5	.173400	.173400
Directly Conn	5		.203000
Sig.		.977	.055

Means and grouping for Response time when **Increasing number of Requests**

Patterns	N	Subset	
		1	2
Directly Conn	50	8452.92	
HubSpoke	50		9532.70
Brokered	50		9636.82
Sig.		1.000	.774

Means and grouping for Thoughput when **Increasing number of Requests**

Patterns	N	Subset	
		1	2
HubSpoke	50	.334320	
Brokered	50	.341320	
Directly Conn	50		.429902
Sig.		.811	1.000

5.8 Performance Comparison of a single ESB and Directly Connected ESBs

In section 5.7, we investigated performance of Directly Connected, Hub-Spoke and Brokered ESB federation pattern. The results were presented in Figure 5.9 to Figure 5.14 and they show that Directly Connected ESB federation pattern outperforms the other patterns according to the criteria investigated. This was expected since this pattern use point-to-point interaction among ESB, thereby speeding communication time between ESBs. Although Directly Connected ESBs pattern provide better performance when compared to other federation pattern, but there was a need to further compare it with a single ESB so as to determine whether having federated ESB was worth in terms of the performance. In order to perform the comparison of Directly Connected ESBs and a single ESB, we had to first find the best ESB when integrated with both UDDI and BPEL engine to support dynamic service discovery and BPEL process execution. Therefore, Section 5.8.1 shows the design and configuration of each ESB when integrated with BPEL engine and UDDI. Then Section 5.8.2 presents the performance evaluation of JBoss, ServiceMix and Mule ESB.

Chapter Four presented the techniques for integrating ESB with each of BPEL engine and UDDI. However, the importance of these two features in addition to ESB would ensure integration

infrastructure that is efficient for dynamic environments like GUISET. Section 5.8.1 presents the integration of each ESB with Apache ODE for service composition and jUDDI for service management and dynamic service discovery.

5.8.1Configuration of each ESB with Apache ODE and jUDDI

In this configuration each ESB is integrated with Apache ODE and jUDDI. Apache ODE execute the LoanBroker BPEL process that coordinate and invoke services that are dynamically discovered from jUDDI. CreditAgencyStore, CreditAgencyBank, Lender and Bank services are published with their QoS attribute in the jUDDI registry. The QoS attribute indicate the availability of the service. The service with the highest availability attribute in each category is selected to participate in the BPEL process.

5.8.1.1Mule ESB

In this section Mule ESB was integrated with both Apache ODE and jUDDI to provide an ESB with dynamic discovery and BPEL process support. LoanBroker service implements the discovery mechanism and the service invocation is defined using BPEL language. Moreover, the BPEL process is executed by the external Apache ODE BPEL engine. Figure 5.15 shows the configuration of Mule integrated with jUDDI and BPEL engine. Service consumer initiates the interaction by sending a SOAP request to the ESB. This request is accepted by HTTP inbound component of Mule ESB. The request is submitted to BPEL LoanBroker service which uses the WSDL specified during the process design and jUDDI client API to discover the actual service address and WSDL interface. Then, the WSDL interface of the actual service was used to invoke the service implementation. CXF WS Client API enabled the interaction between the web service

and BPEL process. Apache ODE was used to execute the BPEL LoanBroker process. Outbound component was used to send the results back to service consumer.

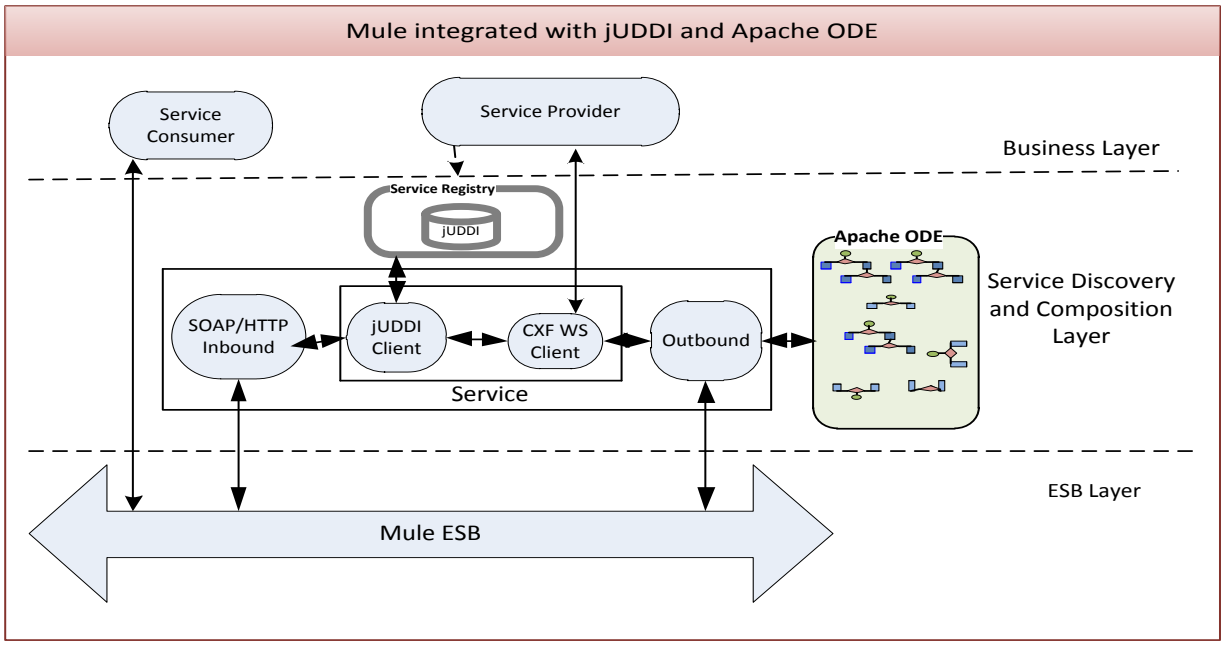

Figure 5.17: Mule Integrated with UDDI and BPEL Engine

5.8.1.2ServiceMix ESB

Apache CXF service engine define the discovery mechanism that is used by the LoanBroker process for discovery of services that participate in the process. The LoanBroker process is also hosted inside an ESB and executed by the external Apache ODE. Service consumer sends the SOAP message to the ESB. Apache CXF BC of ServiceMix accepts the new request. CXF SE then uses jUDDI client to discover the WSDL interfaces for all services needed by the BPEL LoanBroker process. Each service is then invoked as defined by the BPEL process and the results are orchestrated using Apache ODE.

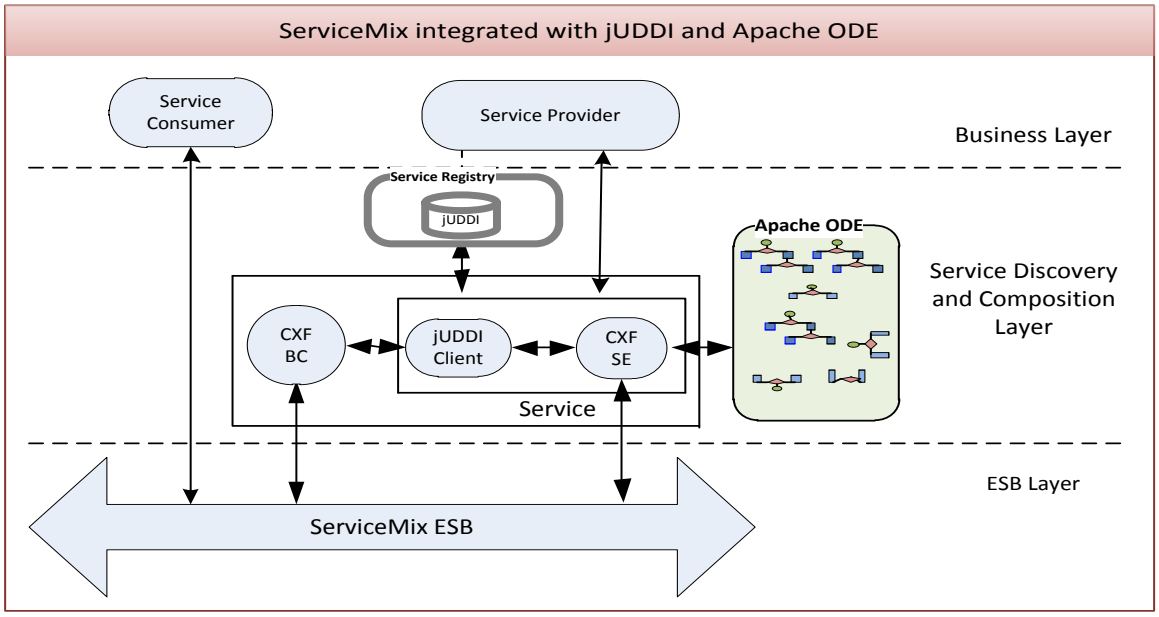

Figure 5.18: ServiceMix ESB integrated with UDDI and BPEL Engine

5.8.1.3JBoss ESB

JBoss ESB uses a SOAP processor for processing SOAP messages that are exchanged between the ESB and Apache ODE during execution. JBoss CXF web service dynamically discovers all services needed by BPEL LoanBroker process.

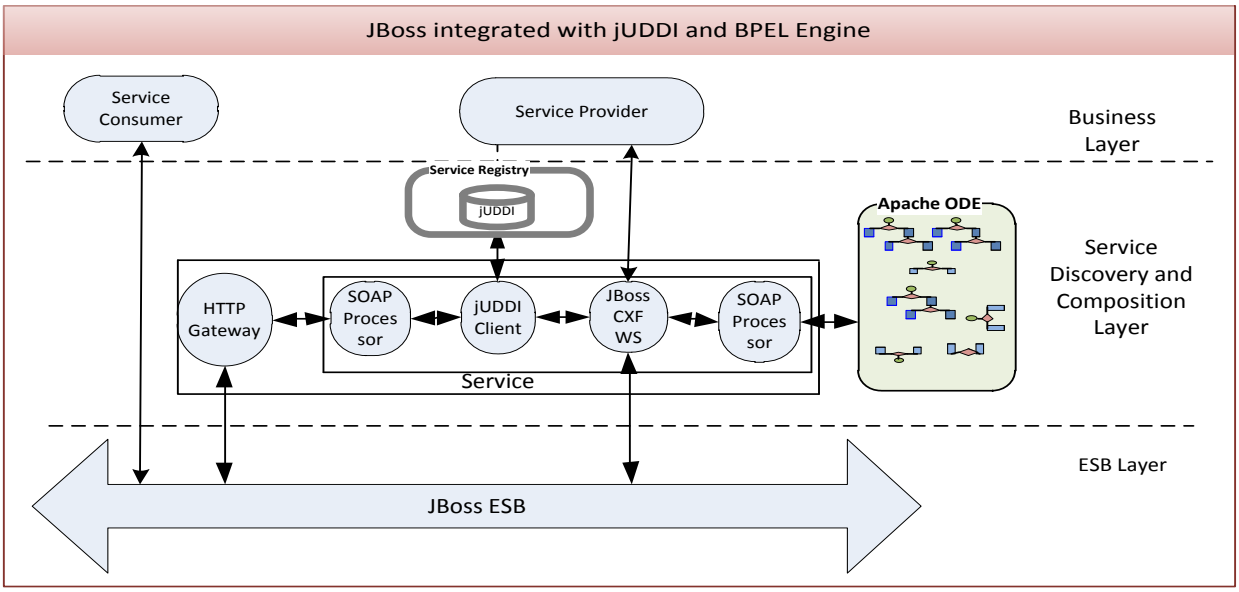

Figure 5.19: JBoss ESB integrated with UDDI and BPEL Engine

5.8.2 Performance evaluation of ESB integrated with UDDI and BPEL Engine

In section 5.8.1, we presented the different configurations of each ESB when integrated with UDDI and BPEL engine. Based on these configurations we carried out different experiments to investigate the scalability of each ESB when performing both dynamic service discovery and executing BPEL process through Apache ODE. Our performance investigation was based on the following three test cases:

- Increasing number of service discovered.
- Increasing number of services published.
- Increasing number of concurrent requests.

Experiment I: Increasing number of service discovered

In this test case, we increased the number of services that were discovered to participate in the process. At Level-0, no discovery took place as all the services participating in the process were directly invoked. Leve-1 the one service was discovered dynamically at runtime by the LoanBroker process, until all the four services were discovered. The results are shown in Figure 5.20 and Figure 5.21. The response in Figure 5.20 increased as the number of services discovered was increasing. There was no exponential growth observed in Figure 5.20, hence all the three ESBs were scalable when increasing number of service discovered. Considering throughput shown in Figure 5.21, we observed the decrease in throughput as the number of services discovered was increasing.

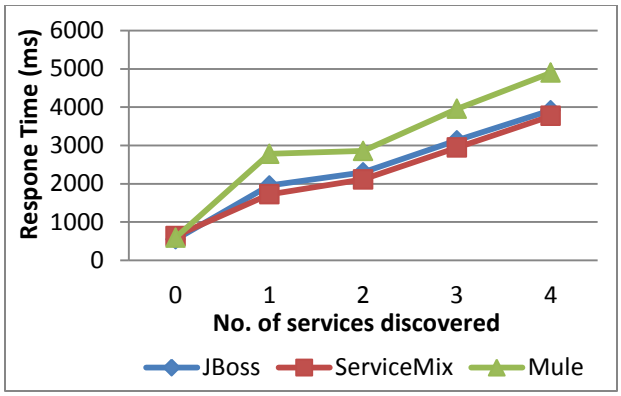

Figure 5.20: Response time vs. No. of services discovered

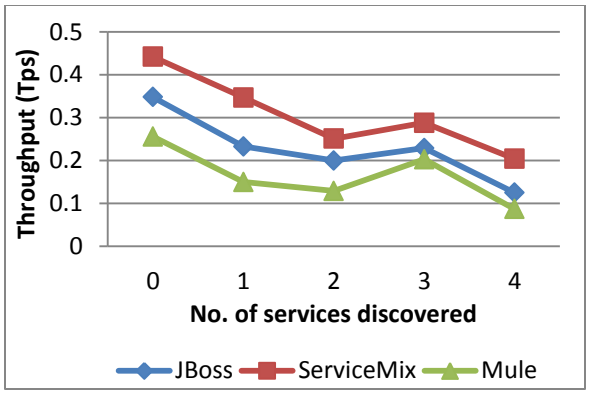

Figure 5.21: Throughput vs. No. of services discovered

Experiment II: Increasing number of services published

The aim of this test case is to investigate the performance of ESBs when increasing number of services published in the jUDDI registry. The services published were increased by registering the replicas of the LoanBroker services. LoanBroker BPEL process select only one service based on its availability as the QoS attribute. Figure 5.22 shows that the response time was increasing as the number of services published was increasing. Considering this test case we can conclude that all ESBs scaled well since there was no quadratic growth in response time as the number of service published increases. Looking at throughput in Figure 5.23 we observed that each ESBs had their best throughput when services published was at 16

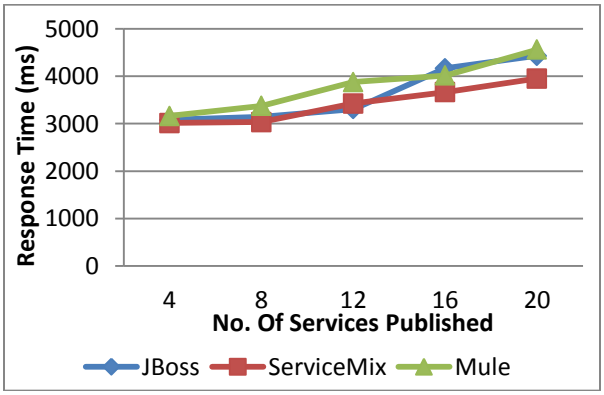

Figure 5.22: Response time vs.no of services published

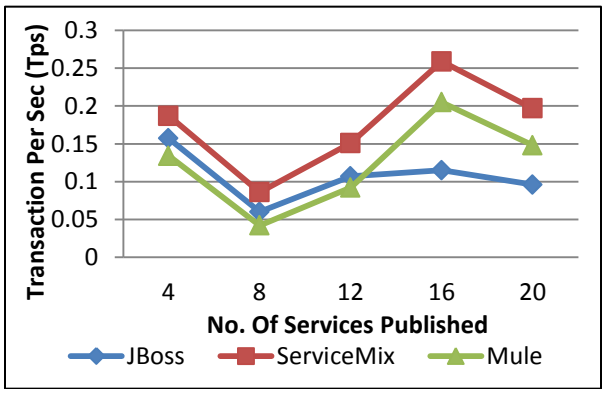

Figure 5.23: Throughput vs. no of services published

Experiment III: Increasing both services published and number of concurrent requests

Increasing number concurrent requests allows us to investigate scalability of each ESB under various loads. In this experiment, LoanBroker BPEL process receives concurrent requests for loan quotes and the clients were measuring response time and throughput. The response time and throughput results obtained are shown in Figure 5.24 and Figure 5.25 respectively. Regarding response time there was not much of the different between ESBs, while for throughput seemed to provide a better throughput then other ESBs.

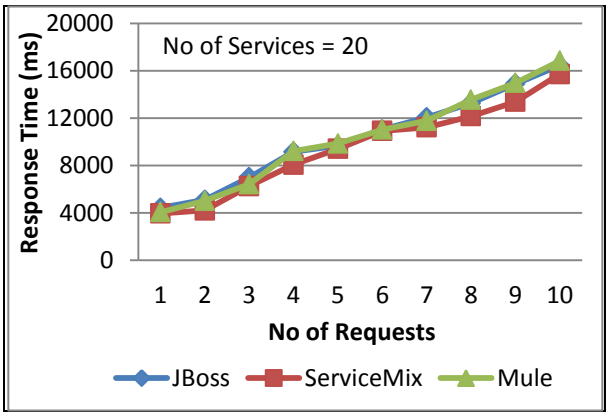

Figure 5.24: Response time vs. No of Requests (Services published = 20)

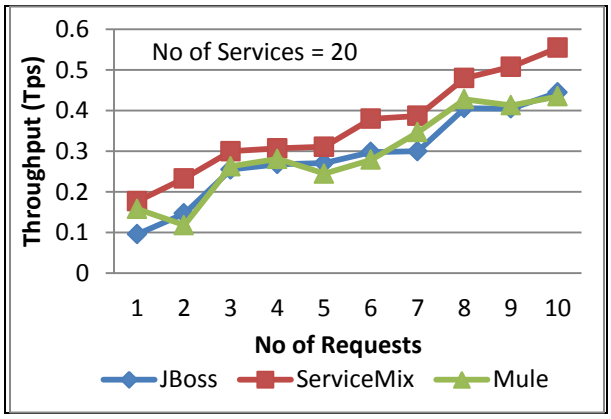

Figure 5.25: Throughput vs. No of Requests (Services published = 20)

5.8.1.3 Results discussion for BPEL with Dynamic Service Discovery

In this experiment, we collected the data that is visually represented by the graphs above, and then we analyzed the data using Univariate ANOVA statistical test. Looking at the P-Values for increasing services discovered, we observed the comparable response time for ServiceMix and JBoss. The significant difference in throughput was observed for all ESBs. The P-Values of increasing service published showed that JBoss had comparable response time with both ServiceMix and Mule. On the other hand, throughput for Mule was comparable to that of JBoss and ServiceMix. Increasing service published with respect to requests showed the response time for ServiceMix was significantly different with that of JBoss and Mule. According to computed

means recorded in Table 7, we can observe that ServiceMix obtained the lowest response time of 2236.40, 3420 and 8883.46 ms in three test cases respectively. In addition ServiceMix achieved the highest throughput of 0.3064, 0.176 and 0.391 Tps in all the test cases. Therefore, we concluded that ServiceMix is the better ESB when considering integration with BPEL engine for business process support and UDDI for dynamic service discovery support. Although JBoss performed better on service discovery, the inclusion of BPEL engine for business processes support showed the drop in performance for JBoss. As it can be observed on the experiment when considering only BPEL service orchestration, JBoss obtained the worse performance which means adding BPEL engine has significant impact (see Chapter Four, Section 4.9.7 and Section 4.9.8).

Table 5.3: P-Values for ESB integrated with UDDI and BPEL Engine

ESBs	**Increasing Services discovered**		**Increasing Services published**		**Increasing services published with Requests**	
	Response Time	Throughput	Response Time	Throughput	Response Time	Throughput
JBoss vs. ServiceMix	**0.698**	0.003	**0.188**	0.015	0.000	0.000
JBoss vs. Mule	0.009	0.011	**0.294**	**0.646**	**0.703**	**0.606**
ServiceMix vs. Mule	0.003	0.00	0.018	**0.058**	0.000	0.000

Table 5.4: Means and grouping of ESBs integrated with UDDI and BPEL Engine

Means and grouping for Response time when **Increasing Services discovered**

ESB	N	Subset	
		1	2
ServiceMix	5	2236.40	
JBoss	5	2369.00	
Mule	5		3017.20
Sig.		.698	1.000

Means and grouping for Throughput when **Increasing Services discovered**

ESB	N	Subset		
		1	2	3
Mule	5	.165000		
JBoss	5		.227000	
ServiceMix	5			.306400
Sig.		1.000	1.000	1.000

Means and grouping forResponse timewhen **Increasing services published**

ESB	N	Subset	
		1	2
ServiceMix	5	3420.00	
JBoss	5	3628.00	3628.00
Mule	5		3800.80
Sig.		.188	.294

Means and grouping forThroughputwhen **Increasing services published**

ESB	N	Subset	
		1	2
JBoss	5	.107000	
Mule	5	.124200	.124200
ServiceMix	5		.176000
Sig.		.646	.058

Means and grouping forResponse timewhen **Increasing number of Requests**

ESB	N	Subset	
		1	2
ServiceMix	50	8883.46	
JBoss	50		9584.72
Mule	50		9665.40
Sig.		1.000	.703

Means and grouping forResponse timewhen **Increasing number of Requests**

ESB	N	Subset	
		1	2
JBoss	50	.314860	
Mule	50	.323240	
ServiceMix	50		.391020
Sig.		.606	1.000

5.8.3 Comparing Performance Results for ServiceMix and Directly Connected ESB

The results presented in Section 5.8.2shows that ServiceMix performed better than the other ESBs when equipped with both dynamic service discovery and BPEL support. Therefore, the

following experiments were carried out to investigate the performance of Directly Connected ESB Federation pattern when compared with ServiceMix. This investigation would assist us to determine whether federating different ESBs and distributing integration tasks to the ESB that has best support has better performance than having a single ESB. The same data obtained when comparing federation patterns (Section 5.7) and single ESBs (Section 5.8.2)was used because these comparisons provide environments that support both dynamic services discovery and BPEL process execution.

The comparison presented in this section is based on the following experiments:

- Increasing number of service discovered.
- Increasing number of services published.
- Increasing number of concurrent requests.

5.8.2.1 Increasing number of services discovered

The aim of this experiment was to investigate the performance of Directly Connected ESBs and ServiceMix when increasing number of services discovered in the UDDI for the composition. The response time and throughput is given in Figure 5.26 and Figure 5.27 respectively. The response time increases as the number of requests increase and we observed that between 0 and 2 services the response time was almost the same. Looking at the throughput, a completely different behavior was observed as increasing number of services discovered. The throughput for Directly Connected ESBs was almost constant between 0 and 3 services discovered and then increases afterwards while for ServiceMix, the throughput decreased and became constant between 2 and 3, then decreased again as more number of services were discovered.

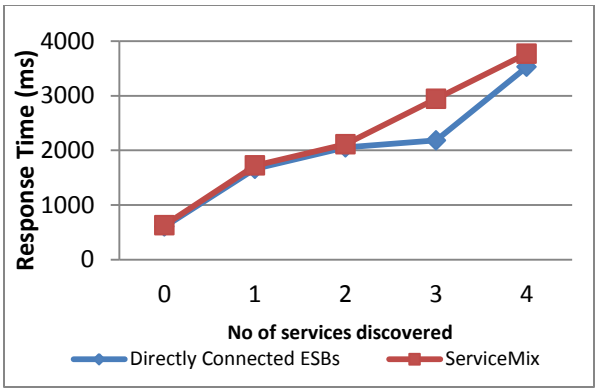

Figure 5.26: Response time vs. no of services discovered

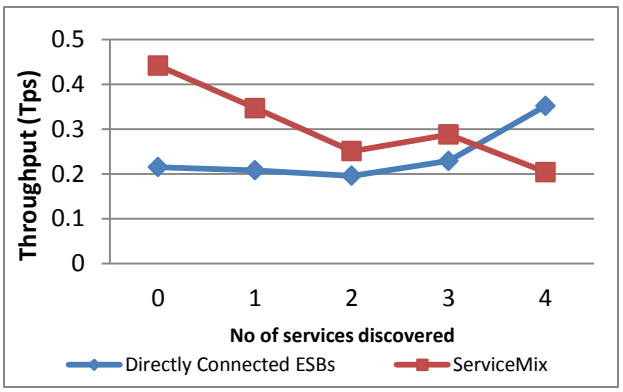

Figure 5.27: Throughput vs. no of services discovered

5.8.2.2Increasing number of services published

The purpose of this experiment was to investigate the response time and transaction time when increasing number of services that are published in the UDDI. Figure 5.28 shows that the response time was comparable for both as it increases with the increase in services that are published. Figure 5.29 presents throughput results as we observed that when we had 16 services published, ServiceMix reached its highest throughput of 0.259 Tps while Directly Connected ESBs pattern reached its lowest throughput of 0.158 Tps.

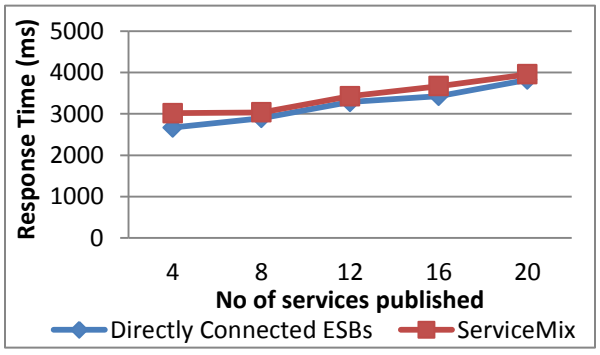

Figure 5.28: Response time vs. no of services published

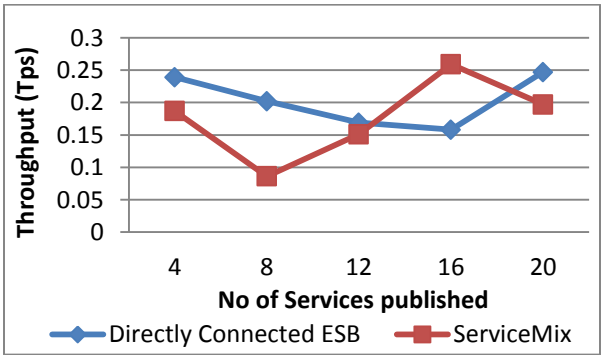

Figure5.29: Throughput vs. no of services published

5.8.2.3 Increasing number of concurrent requests with respect services published

In this experiment, the aim was to investigate the performance when increasing number of concurrent request. Figure 5.30 and Figure 5.31 shows response time and throughput obtained in as the number of requests increase. The response time was comparable as the number of requests increase. We observed that the throughput increases as the number of requests increase.

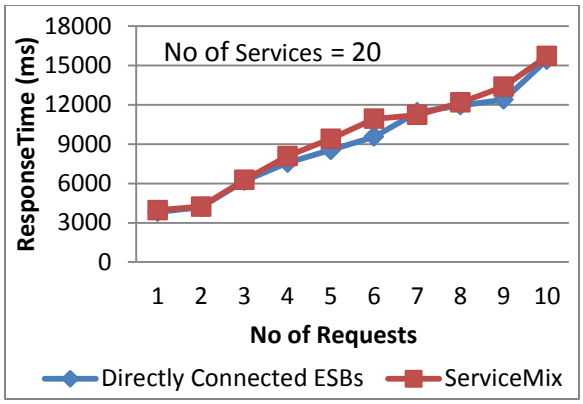

Figure 5.30: Response time vs. increasing no of requests (20 services published)

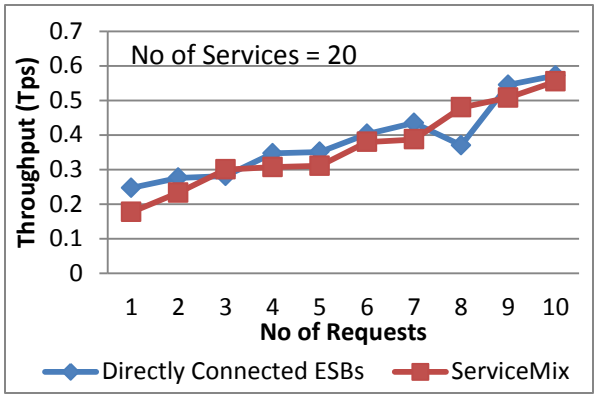

Figure 5.31: Throughput vs. increasing no of requests (20 services published)

5.8.3 Results Discussion for ServiceMix and Directly Connected ESBs

The statistical analysis using Univariate ANOVA method was performed to give meaning to the data collected in the experiments presented above. We looked at the P-Values that were below the set threshold of 0.05. We tested our hypothesis that Directly Connected ESBs would perform better than ServiceMix.

Table 5.5 shows the recorded P-Values for each tested metric. Looking at the increasing number of services discovered both response time and throughput showed that Directly Connected ESBs and ServiceMix had comparable performance. The P-Values for increasing number of services published indicated the difference in response time while the throughput was comparable. Then looking at the P-Values for increasing number of requests with respect to services published, we observed the significant difference in both response time and throughput. Although Directly Connected ESB and ServiceMix achieved considerable performances but in Table 5.6 we noted down the computed means obtained in each metric tested. According to these means Directly Connected ESBs achieved the lower response time of 2005.4 ms, 3221.6 ms and 8452.92 ms in the experiments respectively compared to 2236.4 ms, 3420 ms and 8883.46 ms for ServiceMix. In addition, Directly Connected ESB Federation obtained the highest throughput of 0.203 and

0.43 compared to 0.176 and 0.391 for ServiceMix when increasing number of service published and also when increasing number of requests with respect to number of services.

The fact that ServiceMix was integrated with both UDDI for discovery and BPEL engine for business process support caused a drop in performance for ServiceMix. In addition, we proved in Chapter Four that ServiceMix wasn't the best candidate for dynamic service discovery, so this also contributed to the results obtained by ServiceMix. On the other hand Directly Connected ESBs was made up of three ESBs that work together with each ESB responsible for performing the task that it supports the best. This configuration allows tasks to be distributed among different ESB; in our case JBoss has best support for dynamic service discovery therefore it was integrated with UDDI while ServiceMix was integrated with BPEL engine since it has best support for BPEL processes. So this distribution of responsibilities gave Directly Connected ESB an advantage compared to ServiceMix. In addition, the fact that communication between different ESBs participating in this pattern is direct also contributed to its performance.

Table 5.5: P-Values for Directly Connected ESBs and ServiceMix ESB

Single ESB & Pattern	**Increasing Services discovered**		**Increasing Services published**		**Increasing services published with Requests**	
	Response Time	Throughput	Response Time	Throughput	Response Time	Throughput
DirectlyConn vs. ServiceMix	**0.169**	**0.345**	0.009	**0.492**	0.000	0.000

Table 5.6: Computed mean values for Directly Connected ESBs and ServiceMix ESB

Means for Response time when Increasing Services discovered

Pattern & ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Directly Conn	2005.400	97.478	1734.757	2276.043
ServiceMix	2236.400	97.478	1965.757	2507.043

Means for Throughput when Increasing Services discovered

Pattern & ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Directly Conn	.240	.044	.118	.362
ServiceMix	.306	.044	.184	.428

Means for Response time when Increasing services published

Pattern & ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Directly Conn	3221.600	29.541	3139.581	3303.619
ServiceMix	3420.000	29.541	3337.981	3502.019

Means for Throughput when Increasing services published

Pattern & ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
Directly Conn	.203	.025	.133	.273
ServiceMix	.176	.025	.106	.246

Means for Response time when Increasing Requests

ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
DirrectlyConn	8452.920	67.639	8318.435	8587.405
ServiceMix	8883.460	67.639	8748.975	9017.945

Means for throughput when Increasing Requests

ESB	Mean	Std. Error	95% Confidence Interval	
			Lower Bound	Upper Bound
DirrectlyConn	.430	.006	.417	.443
ServiceMix	.391	.006	.378	.404

5.9 Chapter Summary

This Chapter has presented ESB Federation patterns as they were proposed to reduce some limitations of employing a single ESB in large dynamic SOA environments. These patterns were used to design a communication infrastructure that best support all the GUISET integration requirements. We then presented the implementation of each pattern using LoanBroker scenario in order to illustrate real life settings especially runtime environment, automated service orchestration and dynamic service discovery. The comparative performance evaluation results of Directly Connected, Hub-Spoke and Brokered ESB federation patterns were also presented. The experiments were based on the BPEL defined business process that performs dynamically service discovery. The goal was to investigate the performance of each federation pattern so that one pattern can be recommended for GUISET environment. The results showed that Directly Connected ESB federation pattern outperformed the other patterns. We further compared the performance of Directly Connected ESBs with ServiceMix. The aim of this comparison was to determine whether federating multiple ESBs increase or degrade performance when compared with a single ESB. The results obtained showed Directly Connected ESBs perform better than a single ESB. According to the results presented in this chapter, we can conclude that federating multiple ESBs does not only provide an environment, where all GUISET integration requirement can be supported, but it also improves performance since tasks are distributed to ESBs that have better support for their demands.

CHAPTER SIX

SUMMARY AND FUTURE DIRECTIONS

This chapter gives a summary of the study, highlighting how the goal was achieved, the recommendations based on the results presented from the previous Chapters and gives directions for further work. The limitations of this study were given to highlight the areas of our work that we believe requires improvement in the future.

6.1 Summary

Among enterprise integration approaches that have been developed over the years, the ESB has received a lot of attention and it has become a de facto standard for enterprise integration. This is mainly due to the fact that it provides the infrastructure that paves a way to wide spread adoption of Service Oriented Computing paradigm. The wide adoption of ESBs as the integration solution that provides fast return on investment raise a lot of interest from practitioners and organizations that may want to adopt the use of an ESB. However, with a lot of interest comes a lot of questions on how can this technology be adopted in order to maximally benefit from it. One of these questions found to be of interest to practitioners and organizations is how to choose ESB products that best suit a business's integration needs? When an ESB has been chosen another important question is: how will the chosen ESB be complemented to cater for the integration requirements it may not address? Given the foregoing, this work was a successful attempt to address these questions in the context of GUISET. The goal of this research project was to determine the best strategy that can be used to ensure that the GUISET integration requirements are best supported. The following summarizes what was done towards achieving the goal mentioned above:

i). **This work presented the analysis of ESBs based on multiple conflicting criteria**. Today's large SOA environments demand a lot from their middleware infrastructures in particular ESBs. These demands results in a huge number criteria being used to make decision on which infrastructure to employ in the middleware layer. However these selection criteria might be conflicting due to the fact that different ESB products use different approaches towards achieving a given integration requirement. As a result integration requirements might be best supported by different ESB products. Most previous work focuses on evaluating ESB capabilities, but they do not adequately address the issue of conflicting selection criteria. To address this gap in the GUISET context, this work used a Multi Criteria Decision Analysis Method called Analytical Hierarchy Process to determine which ESB product had better support for which integration requirements. Mule, ServiceMix and JBoss ESBs were evaluated against GUISET integration requirements which are: High Availability, Data Transformation, Content-based Routing, Service Orchestration and Dynamic Service Discovery. Moreover, these GUISET integration requirements were ranked according to their importance in a GUISET-like environment. From this ranking Service Orchestration and Dynamic Service Discovery were ranked as the most important integration requirement for GUISET environment. The analysis results obtained using AHP showed that there is no single ESB that best support all the GUISET integration requirements. For example JBoss performed well when considering dynamic service discover and comparatively worse on other criteria.

ii). **This work presented an approach for integrating ESB with BPEL engine and UDDI to enable dynamic service composition.** The current ESB implementation only support static routing of service communications, this means that messages are transmitted to the specific routing path configured at design time using service configuration file (Yu and Yan, 2011)

(Wu *et al.,* 2008). This static routing is inflexible for service oriented environments where services are dynamically created, deployed and updated. So the decision of which service to select should be taken at execution time based on the recent information. The consequences of this inflexibility are even amplified when multiple services are composed and used in a variety of unplanned-for ways. To address the limitation of the current ESB implementation, this work came up with an approach to integrate ESBs with the BPEL engine and the UDDI. This was more of a by-product to the evaluation of ESB federation patterns which required both dynamic service discovery and service orchestration. In this approach a BPEL defined process dynamically discover service information in the UDDI and coordinate multiple service invocations to give the value-added functionality based on the existing services.

iii). **This study demonstrated the effectiveness in terms of response time and throughput of federating multiple ESBs with each ESB supporting an integration requirement that it best support.** The market trends show that industries are constantly adopting SOA to enable business process automation and integration of heterogeneous IT systems. This adoption of SOA is causing an increase in the number of business-to-business transactions between autonomous SOA deployments. This calls for new and innovative strategies for creating large scale SOA deployments (Callway *et al.,* 2008). Single centralized ESBs are believed to be limited (Kumar *et al.,* 2011). This has led to some practitioners proposing the use of decentralized ESBs. The nature of decentralization is still subject of research but some practitioners have proposed ESB federation. ESB federation has been acknowledged as one of the most promising response to addressing the limitations of a single centralized ESB (Kumar *et al.,* 2011; Baude *et al.,* 2010; Nair, 2009). On the backdrop of the foregoing, this work evaluated federation of ESBs towards

supporting GUISET integration requirements that are best supported by different ESB products. The comprehensive analysis and simulation experiments to investigate performance of Directly Connected, Hub and Spoke and Brokered ESB Federation patterns were presented. The empirical analysis and performance results offer the useful guide for practitioners that seek to adopt a decentralized approach towards building SOA deployment that require integration capabilities that are supported by different ESB products. The results obtained proved that ESB federation provides a better performance compared to single centralized ESB. The reason for this performance improvement is that in the federation different tasks are delegated to the ESBs that best support them.

Based on the results that were presented in this study, we recommend that Directly Connected ESB Federation pattern can be used to design a communication infrastructure for the environment that require integration capabilities that are best supported by different ESB products.

6.2 Limitations and Future Directions

This work determined the best strategy for integrating multiple ESBs in order to ensure that all GUISET integration requirements are best supported. Although this strategy proved to perform better than a single ESB, they are some limitations that are worth mentioning and can be considered in future. These limitations include the fact that we only considered dynamic service discovery and BPEL process support for empirical evaluation. Even though this was well justified by the fact that these two requirements are most important for GUISET environment, in future we would like to also consider high availability, data transformation and message routing for the empirical evaluation. Overall results obtained from this work were based on the simple scenario which can be seen as the approximation of the reality. Therefore, in future, we would

like to observe the behavior of our recommended strategy in real world deployment. We intend to deploy the ESB federation on a dynamic service oriented (like GUISET) withreal service consumers that dynamically discover and compose services.

REFERENCES

Adigun, M., Emuoyibofarhe O., Migiro, S. (2006). Challenges to Access and Opportunity to use SMME enabling Technologies in Africa.In 1st All Africa Technology Diffusion Conference, Johannesburg- South Africa.

AdrioticLogic, (2012).ESB Performance Testing - Round 4.[Available Online: Last Accessed February 2013] http://esbperformance.org/display/comparison/ESB+Performance+Testing+-+Round+4

Ahlberg M., (2010).Enterprise Service Buses: A Comparison Regarding Reliable Message Transfer".Unpublished Masters Dissertation, University of Stockholm, School of Computer Science and Engineering, Royal Institute of Technology, Sweden.

Ahuja, S., & Patel, A. (2011). Enterprise Service Bus: A performance evaluation. In the Scientific Research Communications and Network: Vol. 3(3), (pp. 133-140).

Al-Harbi, K. (2001). Application of the AHP in project management.*International journal of project management*, 19(1), 19-27.

Alphonce, C. B. (1997). Application of the analytic hierarchy process in agriculture in developing countries.*Agricultural systems*, 53(1), 97-112.

Arkin A., Askary S., Bloch B., Curbera F., Goland Y., Kartha N., Liu C., Thatte S., Yendluri P., & Yiu A., (2007). Web Services Business Process Execution Language Version 2.0. [Available Online: Last Accessed February 2013]http://docs.oasis-open.org/wsbpel/2.0/

Apache Software foundation (n.d), "Apache ServiceMix documentation" [Available Online: Last accessed February 2013] http://servicemix.apache.org/

Bai, X., Xie, J., Chen, B., & Xiao, S. (2007). DRESR: Dynamic routing in enterprise service bus. In IEEE International Conference on e-Business Engineering (ICEBE). (pp. 528-531).

Baude, F., Filali, I., Huet, F., Legrand, V., Mathias, E., Merle, P. & Lorre, J. P. (2010).ESB federation for large-scale SOA.In ACM Symposium on Applied Computing (pp. 2459-2466).New York, USA.

Bauler, P., Feltz, F., Biri, N., & Pinheiro, P. (2006).Implementing a Service-Oriented Architecture for Small and Medium Organizations.In Entwicklungsmethod for the information systems and their application(EMISA),(pp. 105-118). Hamburg, Germany

Blum A.,& Fred C., (2004).Representing Web Services Management Information in UDDI.[AvailableOnline : Last accessed February 2013]http://soa.sys-con.com/node/45102.

Brebner, P. (2009). Service-oriented performance modeling the MULE enterprise service bus (ESB) loan broker application. In the 35th IEEE Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 404-411). Lille, France.

Brosey, W., Neal, R.,& Marks, D. (2001).Grand challenges of enterprise integration.In 8th IEEE International Conference onEmerging Technologies and Factory Automation(pp. 221–227).United States of America.

Callaway, R., Viniotis, Y., Rodriguez, A., Brown, K., & Robinson, R. (2008).Enabling federations of enterprise service buses using a distributed service registry.In Second international workshop and Summer School on Service Science, Management and Engineering (SSME), (pp. 2-9). Palermo, Italy.

Cape Clear Software Inc. (2005). Cape Clear Enterprise Service Bus (ESB):How Cape Clear Software applies SOA and Web service principles to deliver a proven ESB solution. Cape Clear Software, United States of America

Chen, I., Ni, G., & Lin, C. (2008). A runtime-adaptable service bus design for telecom operations support systems. IBM Systems Journal, 47(3), 445-456.

Cho, K.(2003). Multicriteria decision methods: an attempt to evaluate and unify. *International Journal of Mathematical and computer modeling*,37(9), 1099-1119.

Clement, L., Hately, A., von Riegen and Rogers, T. (2004).UDDI v 3.0.2 Spec Technical Committee Draft. OASIS

Davis, J. (2009). *Open source SOA*.Greenwick: Manning Publications Co.

Conner, K., DiMaggio, L., Kumar, M., Cunningham, T., Dimaggio, L., & Kumar, M. (2012).*Jboss Esb Beginner's Guide*.Birmingham: Packt Publishing Ltd.

Desmet, S., Volckaert, B., Van Assche, S., Dhoedt, B., & De Turck, F. (2007). Throughput evaluation of different enterprise service bus approaches.In Conf. on Software Eng. Research and Practice (SERP'07), (pp.378- 384).United States of America

Dragicevic, K., Garcés-Erice, L., & Bauer, D. (2010). DISCE: A Declarative Inter-ESB Service-Connectivity Configuration Engine. In IEEE International Conference on Web Services (ICWS), (pp. 489-496).Miami, Florida, United States of America.

Dundek, M. (2010).Service Oriented Architecture and Business Process Management Architecture& Technical Components. In IBM SOA& Business Process Management.

Fakorede, O. (2007). An investigation into the implementation issues and challenges of service oriented architecture. Unpublished Masters Dissertation, Bournemouth University, United Kingdom

García-Jiménez, F., Martínez-Carreras, M., & Gómez-Skarmeta, A. (2010).Evaluating Open Source Enterprise Service Bus. In7th IEEE International Conference on e-Business Engineering (ICEBE), (pp. 284-291). Shanghai, China.

Genender, J. (2006). The Buzz about Enterprise Service Bus (ESB).*In Open Source enterprise Journal: Focusing on open source strategies in the enterprise*,(pp. 34–37).

Gniel, C., and Arnold, D. (2009).Demystifying ESB patterns.In IBM SOA in Action Seminar Series.

Gregory,W.,& Duran, A.,(2001).Scenarios and Acceptance of Forecasts. In the Department of Psychology, New Mexico State University,pp. 519-541. Mexico

Hohpe, G., & Woolf, B. (2004).*Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Indiana: Addison-Wesley Professional.

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., & Hamida, A. (2011). Service-oriented middleware for the Future Internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1), 23-45.

JBoss Community (n.d). JBoss ESB documentation. [Online, Last accessed February 2013] http://www.jboss.org/jbossesb/docs

Jongtaveesataporn, A., & Takada, S. (2010). Enhancing enterprise service bus capability for load balancing.*Journal World Scientific and Engineering Academy and Society (WSEAS) Transactions on Computers*, 9(3), 299-308.

Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C.,& Verschueren, P. (2004). *Patterns: Implementing an SOA using an enterprise service bus*. New York City: IBMCorporation, International Technical Support Organization.

Khan, F., Bashir, S., Javed, M., Khan, A., & Khiyal, M. (2010). QoS Based Dynamic Web Services Composition & Execution.*International Journal of Computer Science and Information Security*, 7(2), 147-152.

Kumar, S. Britto. R., & Rabara, S. A. (2011).Multi-level Security for Integrated Financial Mobile Web Services using Federated ESB. *International Journal of Computer Applications*, 32(10), 39-45.

Kusak, D. (2010). Comparison of Enterprise Application Integration Platforms.Unpublished MastersDissertation, Charles University in Prague, Czech

Kruessmann, T., Koschel, A., Murphy, M., Trenaman, A., & Astrova, I. (2009). High availability: Evaluating open source enterprise service buses. In31st IEEE International Conference on Information Technology Interfaces (pp. 615-620). Croatia

Lima, B.(2011). MBC-Mobile Business Collaboration.Unpublished Mastersdissertation, Universidade Tecnica de Lisboa, 2011, Portugal

Linthicum, D. (2006). Ws-bpel 2.0: Not backward compatible?. [Available Online,Last accessed February 2013] http://webservices.sys-con.com/read/291050.htm

Luo, M., Goldshlager, B., and Zhang, L. (2005).Designing and implementing Enterprise Service Bus(ESB) and SOA solutions Tutorial. In IEEE International Conference on Services Computing(SCC), Orland, FL, United States of America.

Mareno, M.,& Raffeala, M. (2010).QoS Analysis for Web Service Applications: a Survey on Performance Oriented approaches from the Architecture Point of View. In Technical Report UBLCS-2010-05, Italy

Menascé, D. A., & Almeida, V. (2002). *Capacity Planning for Web Services: metrics, models, and methods* (p. 133- 140). New Jersey: Prentice Hall.

Menge, F. (2007). Enterprise Service Bus. In Free and Open Source Software Conference.United State of America.

MuleSoft(n.d).Mule ESB documentation. [Available Online, Last accessed February 2013] http://www.mulesoft.org/documentation/dashboard.action

Mulik, S. (2009).Using Enterprise Service Bus (ESB) for connecting corporate functions and shared services with business divisions in a large enterprise.InIEEE Asia-Pacific Conference on Services Computing (APSCC), (pp. 430-434). China.

Nair S. (2009). Why the Time Has Come for Federated ESBs. In SOA Security [Available Online: Last Accessed in February 2013]

http://www.ebizq.net/topics/soa_security/features/10915.html

Ortiz, S. (2007). Getting on board the enterprise service bus. Computer Archives, 40(4), 15-17.

OW2 Middleware consortium (n.d).People in Petals Forum. [Available Online: Last Accessed February2013] http://forum.petalslink.com/template/NamlServlet.jtp?macro=app_people&node=1863229

Perera, A. (2008). WSO2 Enterprise Service Bus (ESB) Performance Testing Round 3. Oxygen tank [Available Online: Last Accessed in February 2013] http://wso2.org/library/3740

Papazoglou, M., &Heuvel, W. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*,16(3), 389-415.

Papazoglou, P. (2008). Web Services: Principles and Technology, 1st Edition, © Pearson Education Limited, Netherlands.

Patil, N., & Gopal, A. (2012). Enhancing UDDI registry for storing Qos in tModel for discovering web services.*International Journal of Computer Science and Application*, 9(3)

Patton, M. (2001).*Qualitative research & evaluation methods*. London: Sage Publications Incorporated.

Rajendran, T., & Balasubramanie, P. (2009).Analysis on the Study of QoS-Aware Web Services Discovery.*Journal of Computing*, 1(1), 119-130

Saaty, T., & Shih, H. S. (2009). Structures in decision making: On the subjective geometry of hierarchies and networks. *European Journal of Operational Research*, 199(3), 867-872.

Saaty, T. (2007).Decision making, scaling, and number crunching. *Decision Sciences*, 20(2), 404-409.

Serhani, M., Nabeel, A., &Abdelghani, B. (2010).Enterprise services (business) collaboration using portal and SOA-based semantics.In 4th IEEE International Conference onDigital Ecosystems and Technologies (DEST) (pp. 450- 455).United Arab Emirates, Dubai

Shehab, E., Sharp,W., Supramaniam,L., & Spedding, T. (2004) Enterprise resource planning: An integrative review.*Business Process Management Journal*, 10 (4), 359-386.

Siddiqui, Z., Abdullah, A. H., Khan, M. K., & Alghathbar, K. (2011).Analysis of enterprise service buses based on information security, interoperability and high-availability using Analytical Hierarchy Process (AHP) method.*International Journal of the Physical Sciences*, 6(1), pp. 35-42.

Soomro, T., & Awan, A. (2012).Challenges and Future of Enterprise Application Integration.*International Journal of Computer Applications*, 42(7), 42-45.

Steele, K., Carmel, Y., Cross, J., & Wilcox, C. (2009). Uses and misuses of multicriteria decision analysis (MCDA) in environmental decision making. *Risk analysis*, 29(1), 26-33.

Ten-Hove, R., & Walker, P. (2005).Java Business Integration (JBI) 1.0.Java Specification Request, 208.

Tiwari, V., Dagdee, N., Tiwari, A., & Dixit, D. (2012).Extended SOA to Enable Web Service Discovery on Non Functional Parameters.*International Journal of Electronics Communication and Computer Engineering*, 3(2), 97-100.

Themistocleous, M., Irani, Z. & O'Keefe, R. (2004).ERP and application integration: Exploratory survey.*Business Process Management Journal*, 7(3), 195-204.

Triantaphyllou, E., & Mann, S. (1995). Using the analytic hierarchy process for decision making in engineering applications: some challenges. *International Journal of Industrial Engineering: Applications and Practice*, 2(1), 35-44.

Ueno, K., & Tatsubori, M. (2006).Early capacity testing of an enterprise service bus. In IEEE International Conference onWeb Services(pp. 709-716). Chicago, USA

Vollmer, K., Gilpin M., and Rose S.(2011).The Forrester Wave™: Enterprise Service Bus, Q2 2011. Forrester Research, Inc, 10.

Wu, B., Liu, S., & Wu, L. (2008).Dynamic reliable service routing in enterprise service bus.In IEEE Asia-Pacific Services Computing Conference (APSCC) (pp. 349-354).

Wu, B., & Wu, X. (2010).A QoS-aware Method for Web Services Discovery.*Journal of Geographic Information System*, 2 (1), 40-44.

Yu, D., & Yan, D. (2011). Towards the integration of Enterprise Service Bus with UDDI server: A case study. In IEEEInternational Conference on System Science and Engineering (ICSSE), (pp. 28-31).Macau

Yin, R. (2008).*Case study research: Design and methods*.London: Sage PublicationsIncorporated.

Zdravković,M., Trajanović, M., & Manić M., (2007).SOA-based approach to the Enterprise resource planning implementation in Small Enterprises.In Series of Mechanical Engineering, 5 (1), 97– 104.

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), 311-327.

Ziyaeva, G., Choi, E., & Min, D. (2008). Content-based intelligent routing and message processing in enterprise service bus. In IEEE International Conference on Convergence and Hybrid Information Technology (ICHIT). (pp. 245-249). Washington DC, United States of America

Appendix A: ESB Configuration and Installations

The installation and configuration steps listed below are for Windows operation system

ServiceMix ESB

Installation

- Java and Maven need to be installed at first
    - Java version 1.7 was installed. The installation can be found at http://docs.oracle.com/javase/7/docs/webnotes/install/windows/jdk-installation-windows.html
    - Maven version 3.0.3 was installed. Instruction cab be found at http://maven.apache.org/download.cgi#Installation
- Download Apache ServiceMix ESB 3.4.0 (.zip distribution) from http://servicemix.apache.org/downloads/servicemix-3.4.0
- Unpack the .zip file anywhere in the system. This folder we will refer to it as <*inst_dir*>
- The is no need for SERVICEMIX_HOME

Running

- Use command prompt, Go to <*inst_dir*>*/bin* and type servicemix.bat
- ServiceMix ESB will start and automatically deploy all the default JBI components
- Item logs can be found in <*inst_dir*>*/data/log/servimix*.log

Building and Deploying Projects

- Ensure that ServiceMix is running, use command prompt to navigate to <*proj_dir*> and type *mvn install*. This should install the project, if successful it should end with a message "BUILD SUCCESSFUL"
- Then copy <*proj_dir*>*/projectName-cxf-sa-1.0 .zip* to <*inst_dir*>*/hotdeploy*. The project should deploy and ServiceMix server should indicate deploy successfully

Tuning

- The default java memory was too low for our experiments, to increase memory – Edit <*inst_dir*>*/conf/*servicemix.xml file. At the beginning add these two lines;
    - JAVA_MIN_MEM = 512M
    - JAVA_MAX_MEM = 1024M
- The default maximum PoolSize of ServiceMix is 32 which is too small. To increase HTTP gateway we had to edit <*inst_dir*>*/config/*servicemix.properties by modifying servicemix.maximumPoolSize = 64

Mule ESB

Installation

- Mule also needs apache maven installed
- Download Mule Community distribution from http://www.mulesoft.org/download-mule-esb-community-edition
- Select Mule ESB full distribution standalone server
- Unpack the downloaded file to somewhere in the system
- Set MULE_HOME to mule path (for example: *C://MuleESB/mule-standalone-3.2.0*)

Start Mule

- Use command prompt, go to *<inst_dir>/bin* and type *mule.bat*. Mule would be started.

JBoss ESB

Installation

- Prerequisite: jdk6 or newer should be installed
- Install Apache ant
- Download jbossesb-server-4.9 .zip folder and place it anywhere in the system
- No JBOSS_HOME is needed

Running JBoss

- Use command prompt to navigate to *<inst_dir>/bin* and type *run.bat*. JBoss will start
- Admin console available at http://localhost:8080/admin-console- username=admin, password=admin
- Logs are available in the folder *<inst_dir>/server/default/log*

Building projects

- Edit file*<JBossProject>/conf/*base-biuld.xml. Set server installation path to jbossesb server by editing*<property name= "product.dir" location = "/jbossesb-server-4.9"/>*
- Navigate to *<JBossProject>* ant type *ant deploy*
- JBoss ESB server log should show message that a new project successfully deployed

Appendix B: Source Code

BPEL LoanBroker Process Implemented for ESB Federation

```
<!-- DirectESB-FederationDSD-1 BPEL Process [Generated by the Eclipse BPEL
Designer] -->
<!-- Date: Mon Dec 17 19:26:28 CAT 2012 -->
<bpel:processname="DirectESB-FederationDSD-1"
targetNamespace="http://DirectESB-FederationDSD-1.process"
suppressJoinFailure="yes"
xmlns:tns="http://DirectESB-FederationDSD-1.process"
xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
xmlns:ns="http://webservice_creditAgencyStore/creditAgencyDirectNoDisc"xmlns:
ns0="http://servicemix.apache.org/examples"xmlns:ns1="http://webservice_BankS
erviceHR/BankServiceHR"xmlns:ns2="http://webservice_BankServiceLR/BankService
LR">

<!-- Import the client WSDL -->
<bpel:importnamespace="http://webservice_BankServiceLR/BankServiceLR"location
="MuleBankLR.wsdl"importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import
>
<bpel:importnamespace="http://webservice_BankServiceHR/BankServiceHR"location
="MuleBankHR.wsdl"importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import
>
<bpel:importnamespace="http://servicemix.apache.org/examples"location="SMXLen
derService.wsdl"importType="http://schemas.xmlsoap.org/wsdl/"></bpel:import>
<bpel:importnamespace="http://webservice_creditAgencyStore/creditAgencyDirect
NoDisc"location="JBossCreditAgency.wsdl"importType="http://schemas.xmlsoap.or
g/wsdl/"></bpel:import>
<bpel:importlocation="DirectESB-FederationDSD-
1Artifacts.wsdl"namespace="http://DirectESB-FederationDSD-1.process"
      importType="http://schemas.xmlsoap.org/wsdl/"/>

<!-- ================================================================= -->
<!-- PARTNERLINKS                                                      -->
<!-- List of services participating in this BPEL process               -->
<!-- ================================================================= -->
<bpel:partnerLinks>
<!-- The 'client' role represents the requester of this service. -->
<bpel:partnerLinkname="client"
partnerLinkType="tns:DirectESB-FederationDSD-1"
myRole="DirectESB-FederationDSD-1Provider"
/>
<bpel:partnerLinkname="JBossCreditAgencyPL"partnerLinkType="tns:JBossCreditAg
encyPLT"partnerRole="JBossCreditAgencyRloe"></bpel:partnerLink>
<bpel:partnerLinkname="SMXLenderPL"partnerLinkType="tns:SMXLenderPLT"partnerR
ole="SMXLenderRole"></bpel:partnerLink>
<bpel:partnerLinkname="MuleBankHRPL"partnerLinkType="tns:MuleBankHRPLT"partne
rRole="MuleBankHRRole"></bpel:partnerLink>
<bpel:partnerLinkname="MuleBankLRPL"partnerLinkType="tns:MuleBankLRPLT"partne
rRole="MuleBankLRRole"></bpel:partnerLink>
</bpel:partnerLinks>

<!-- ================================================================= -->
<!-- VARIABLES                                                         -->
```

```
<!-- List of messages and XML documents used within this BPEL process  -->
<!-- ================================================================= -->
<bpel:variables>
<!-- Reference to the message passed as input during initiation -->
<bpel:variablename="input"
messageType="tns:DirectESB-FederationDSD-1RequestMessage"/>

<!--
          Reference to the message that will be returned to the requester
          -->
<bpel:variablename="output"
messageType="tns:DirectESB-FederationDSD-1ResponseMessage"/>
<bpel:variablename="JBossCreditAgencyPLResponse"messageType="ns:creditAgencyD
irectNoDisc_getCreditHistoryResponse"></bpel:variable>
<bpel:variablename="JBossCreditAgencyPLRequest"messageType="ns:creditAgencyDi
rectNoDisc_getCreditHistory"></bpel:variable>
<bpel:variablename="SMXLenderPLResponse"messageType="ns0:selectLendersRespons
e"></bpel:variable>
<bpel:variablename="SMXLenderPLRequest"messageType="ns0:selectLendersRequest"
></bpel:variable>
<bpel:variablename="MuleBankHRPLResponse"messageType="ns1:BankServiceHR_getBa
nkServiceHRRateResponse"></bpel:variable>
<bpel:variablename="MuleBankHRPLRequest"messageType="ns1:BankServiceHR_getBan
kServiceHRRate"></bpel:variable>
<bpel:variablename="MuleBankHRPLResponse1"messageType="ns1:BankServiceHR_getB
ankServiceHRQouteResponse"></bpel:variable>
<bpel:variablename="MuleBankHRPLRequest1"messageType="ns1:BankServiceHR_getBa
nkServiceHRQoute"></bpel:variable>
<bpel:variablename="MuleBankLRPLResponse"messageType="ns2:BankServiceLR_getBa
nkServiceLRRateResponse"></bpel:variable>
<bpel:variablename="MuleBankLRPLRequest"messageType="ns2:BankServiceLR_getBan
kServiceLRRate"></bpel:variable>
<bpel:variablename="MuleBankLRPLResponse1"messageType="ns2:BankServiceLR_getB
ankServiceLRQouteResponse"></bpel:variable>
<bpel:variablename="MuleBankLRPLRequest1"messageType="ns2:BankServiceLR_getBa
nkServiceLRQoute"></bpel:variable>
</bpel:variables>

<!-- ================================================================= -->
<!-- ORCHESTRATION LOGIC                                               -->
<!-- Set of activities coordinating the flow of messages across the    -->
<!-- services integrated within this business process                  -->
<!-- ================================================================= -->
<bpel:sequencename="main">

<!-- Receive input from requester.
             Note: This maps to operation defined in DirectESB-FederationDSD-
1.wsdl
             -->
<bpel:receivename="receiveInput"partnerLink="client"
portType="tns:DirectESB-FederationDSD-1"
operation="process"variable="input"
createInstance="yes"/>

<!-- Generate reply to synchronous request -->
<bpel:assignvalidate="no"name="AssignToCeditAgency">
<bpel:copy>
```

```
<bpel:from><bpel:literal><tns:getCreditHistoryxmlns:tns="http://webservice_creditAgencyStore/creditAgencyDirectNoDisc"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<amount>amount</amount>
</tns:getCreditHistory>
</bpel:literal></bpel:from>
<bpel:tovariable="JBossCreditAgencyPLRequest"part="getCreditHistory"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:input]]></bpel:query>
</bpel:from>
<bpel:topart="getCreditHistory"variable="JBossCreditAgencyPLRequest">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[amount]]></bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invokename="InvokeCreditAgency"partnerLink="JBossCreditAgencyPL"operation="getCreditHistory"portType="ns:creditAgencyDirectNoDisc"inputVariable="JBossCreditAgencyPLRequest"outputVariable="JBossCreditAgencyPLResponse"></bpel:invoke>
<bpel:assignvalidate="no"name="AssignToLender">
<bpel:copy>
<bpel:from><bpel:literal><ns:selectLendersxmlns:ns="http://servicemix.apache.org/examples"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ns:amount>ns:amount</ns:amount>
<ns:creditHistory>ns:creditHistory</ns:creditHistory>
</ns:selectLenders>
</bpel:literal></bpel:from>
<bpel:tovariable="SMXLenderPLRequest"part="parameters"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[tns:input]]></bpel:query>
</bpel:from>
<bpel:topart="parameters"variable="SMXLenderPLRequest">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[ns0:amount]]></bpel:query>
</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="getCreditHistoryResponse"variable="JBossCreditAgencyPLResponse">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[return]]></bpel:query>
</bpel:from>
<bpel:topart="parameters"variable="SMXLenderPLRequest">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![CDATA[ns0:creditHistory]]></bpel:query>
</bpel:to>
</bpel:copy>

</bpel:assign>
```

```
<bpel:invokename="InvokeLenders"partnerLink="SMXLenderPL"operation="selectLen
ders"portType="ns0:Lenders"inputVariable="SMXLenderPLRequest"outputVariable="
SMXLenderPLResponse"></bpel:invoke>
<bpel:ifname="CheckLenderRating">
<bpel:condition><![CDATA[$SMXLenderPLResponse.parameters/ns1:return=
"BankServiceHR"]]></bpel:condition>
<bpel:sequence>
<bpel:assignvalidate="no"name="AssignToBankRate">
<bpel:copy>
<bpel:from><bpel:literal><tns:getBankServiceHRRatexmlns:tns="http://webservic
e_BankServiceHR/BankServiceHR"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<amount>amount</amount>
</tns:getBankServiceHRRate>
</bpel:literal></bpel:from>
<bpel:tovariable="MuleBankHRPLRequest"part="getBankServiceHRRate"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[tns:input]]></bpel:query>
</bpel:from>
<bpel:topart="getBankServiceHRRate"variable="MuleBankHRPLRequest">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[amount]]></bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invokename="InvokeBankHRRate"partnerLink="MuleBankHRPL"operation="getBa
nkServiceHRRate"portType="ns1:BankServiceHR"inputVariable="MuleBankHRPLReques
t"outputVariable="MuleBankHRPLResponse"></bpel:invoke>
<bpel:assignvalidate="no"name="AssignToHRLoanQoute">
<bpel:copy>
<bpel:from><bpel:literal><tns:getBankServiceHRQoutexmlns:tns="http://webservi
ce_BankServiceHR/BankServiceHR"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<rate>rate</rate>
<amount>amount</amount>
</tns:getBankServiceHRQoute>
</bpel:literal></bpel:from>
<bpel:tovariable="MuleBankHRPLRequest1"part="getBankServiceHRQoute"></bpel:to
>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[tns:input]]></bpel:query>
</bpel:from>
<bpel:topart="getBankServiceHRQoute"variable="MuleBankHRPLRequest1">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[amount]]></bpel:query>
</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="getBankServiceHRRateResponse"variable="MuleBankHRPLResponse">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[return]]></bpel:query>
```

```
</bpel:from>
<bpel:topart="getBankServiceHRQoute"variable="MuleBankHRPLRequest1">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[rate]]></bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invokename="InvokeHRLaonQouteResults"partnerLink="MuleBankHRPL"operatio
n="getBankServiceHRQoute"portType="ns1:BankServiceHR"inputVariable="MuleBankH
RPLRequest1"outputVariable="MuleBankHRPLResponse1"></bpel:invoke>
<bpel:assignvalidate="no"name="AssignToHROutput">
<bpel:copy>
<bpel:from><bpel:literal><tns:DirectESB-FederationDSD-
1Responsexmlns:tns="http://DirectESB-FederationDSD-
1.process"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tns:result>tns:result</tns:result>
</tns:DirectESB-FederationDSD-1Response>
</bpel:literal></bpel:from>
<bpel:tovariable="output"part="payload"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="getBankServiceHRQouteResponse"variable="MuleBankHRPLResponse1
">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[return]]></bpel:query>
</bpel:from>
<bpel:topart="payload"variable="output">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[tns:result]]></bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:replyname="ReplyHRLoanQouteResponse"partnerLink="client"operation="proc
ess"portType="tns:DirectESB-FederationDSD-1"variable="output"></bpel:reply>
</bpel:sequence>
<bpel:else>
<bpel:sequence>
<bpel:assignvalidate="no"name="AssignToBankRate">
<bpel:copy>
<bpel:from><bpel:literal><tns:getBankServiceLRRatexmlns:tns="http://webservic
e_BankServiceLR/BankServiceLR"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<amount>amount</amount>
</tns:getBankServiceLRRate>
</bpel:literal></bpel:from>
<bpel:tovariable="MuleBankLRPLRequest"part="getBankServiceLRRate"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[tns:input]]></bpel:query>
</bpel:from>
<bpel:topart="getBankServiceLRRate"variable="MuleBankLRPLRequest">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[amount]]></bpel:query>
</bpel:to>
</bpel:copy>
```

```
</bpel:assign>
<bpel:invokename="InvokeBankLRRate"partnerLink="MuleBankLRPL"operation="getBa
nkServiceLRRate"portType="ns2:BankServiceLR"inputVariable="MuleBankLRPLReques
t"outputVariable="MuleBankLRPLResponse"></bpel:invoke>
<bpel:assignvalidate="no"name="AssignToLRLoanQoute">
<bpel:copy>
<bpel:from><bpel:literal><tns:getBankServiceLRQoutexmlns:tns="http://webservi
ce_BankServiceLR/BankServiceLR"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
<rate>rate</rate>
<amount>amount</amount>
</tns:getBankServiceLRQoute>
</bpel:literal></bpel:from>
<bpel:tovariable="MuleBankLRPLRequest1"part="getBankServiceLRQoute"></bpel:to
>
</bpel:copy>
<bpel:copy>
<bpel:frompart="getBankServiceLRRateResponse"variable="MuleBankLRPLResponse">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[return]]>
</bpel:query>
</bpel:from>
<bpel:topart="getBankServiceLRQoute"variable="MuleBankLRPLRequest1">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[rate]]>
</bpel:query>
</bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="payload"variable="input">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[tns:input]]>
</bpel:query>
</bpel:from>
<bpel:topart="getBankServiceLRQoute"variable="MuleBankLRPLRequest1">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
<![CDATA[amount]]>
</bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:invokename="InvokeLRLaonQouteResults"partnerLink="MuleBankLRPL"operatio
n="getBankServiceLRQoute"portType="ns2:BankServiceLR"inputVariable="MuleBankL
RPLRequest1"outputVariable="MuleBankLRPLResponse1"></bpel:invoke>
<bpel:assignvalidate="no"name="AssignToLROutput">
<bpel:copy>
<bpel:from><bpel:literal><tns:DirectESB-FederationDSD-
1Responsexmlns:tns="http://DirectESB-FederationDSD-
1.process"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tns:result>tns:result</tns:result>
</tns:DirectESB-FederationDSD-1Response>
</bpel:literal></bpel:from>
<bpel:tovariable="output"part="payload"></bpel:to>
</bpel:copy>
<bpel:copy>
<bpel:frompart="getBankServiceLRQouteResponse"variable="MuleBankLRPLResponse1
">
```

```
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[return]]></bpel:query>
</bpel:from>
<bpel:topart="payload"variable="output">
<bpel:queryqueryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0"><![
CDATA[tns:result]]></bpel:query>
</bpel:to>
</bpel:copy>
</bpel:assign>
<bpel:replyname="ReplyLRLoanQouteResponse"partnerLink="client"portType="tns:D
irectESB-FederationDSD-1"operation="process"variable="output"/>
</bpel:sequence>
</bpel:else>
</bpel:if>

</bpel:sequence>
</bpel:process>
```

Dynamic Service Discovery Mechanism

```
public class DynamicInvoker {
        private static UDDISecurityPortType security = null;
private static UDDIInquiryPortType inquiry = null;
private AuthToken rootAuthToken;
private String accesspoint;
private String serviceName;
private String serviceEndpoint;
private String serviceKey;
        public DynamicInvoker(){
                try {
                        String clazz =
UDDIClientContainer.getUDDIClerkManager(null).getClientConfig().getUDDINode("default").getProxyTransport(
);
                        Class<?> transportClass = ClassUtil.forName(clazz, Transport.class);
                        if (transportClass!=null) {
                                Transport transport = (Transport)
transportClass.getConstructor(String.class).newInstance("default");
                                security = transport.getUDDISecurityService();
                                inquiry = transport.getUDDIInquiryService();
                        }
                } catch (Exception e) {
e.printStackTrace();
        }
    }
public String getAuthToken(){
try {
        // Setting up the values to get an authentication token for the 'root' user ('root' user has admin privileges
        // and can save other publishers).
        GetAuthToken getAuthTokenRoot = new GetAuthToken();
getAuthTokenRoot.setUserID("root");
getAuthTokenRoot.setCred("root");
```

```
          // Making API call that retrieves the authentication token for the 'root' user.
rootAuthToken = security.getAuthToken(getAuthTokenRoot);
return rootAuthToken.getAuthInfo();
        } catch (Exception e) {
e.printStackTrace();
return null;
        }
     }
public ServiceList findBusServiceList(String serviceName){

try{
FindService  findservice = new FindService();
findservice.setAuthInfo(getAuthToken());
        FindQualifiers q = new FindQualifiers();
q.getFindQualifier().add("ApproximateMatch");
findservice.setFindQualifiers(q);
        Name sName = new Name();
sName.setValue(serviceName);
findservice.getName().add(sName);
        ServiceList services = inquiry.findService(findservice);
return services;
       }
catch (Exception e){
e.printStackTrace();
return null;
       }

   }
public BusinessService GetBusServiceDetails(ServiceList services, int Serviceindex){
           try{

        String servicekey = services.getServiceInfos().getServiceInfo().get(Serviceindex).getServiceKey();

        GetServiceDetail serviceDetails = new GetServiceDetail();
serviceDetails.setAuthInfo(getAuthToken());
serviceDetails.getServiceKey().add(servicekey);
        ServiceDetail serviceDet = inquiry.getServiceDetail(serviceDetails);

        BusinessService businessService = serviceDet.getBusinessService().get(0);
           return businessService;
       }
           catch(Exception e){
           e.printStackTrace();
           return null;
           }
   }

public int ServiceIndex(ServiceList services){
int maximumQoS = 70;
int servIndex = 0;
for(int i = 0; i < services.getListDescription().getActualCount(); i++){
         BusinessService service = GetBusServiceDetails(services, i);
         TModel tmodel = getTmodel(service);
         //String QosName = getQoSName(tmodel);
         String QoSValue = getQoSValue(tmodel);
```

```
int actualQosValue = Integer.parseInt(QoSValue);
if (actualQosValue>=maximumQoS){
maximumQoS = actualQosValue;
servIndex = i;
        }
     }
return servIndex;
   }

public String getAccesspoint(BusinessService service){
          serviceKey =  service.getServiceKey();

          try
          {
        GetServiceDetail getServiceDetail=new GetServiceDetail();
getServiceDetail.setAuthInfo(getAuthToken());
getServiceDetail.getServiceKey().add(serviceKey);

        ServiceDetail serviceDetail=inquiry.getServiceDetail(getServiceDetail);

        BusinessService businessservice=serviceDetail.getBusinessService().get(0);
serviceName = businessservice.getName().get(0).getValue();
        //wsdlTA.append("fetched service name:"+serviceName+"\n");

        String bindingkey = businessservice.getBindingTemplates().getBindingTemplate().get(0).getBindingKey();
        //wsdlTA.append("fetched binding key:"+bindingkey+"\n");

        GetBindingDetail gbd = new GetBindingDetail();
gbd.setAuthInfo(rootAuthToken.getAuthInfo());
gbd.getBindingKey().add(bindingkey);
        BindingDetail bindingdetail=inquiry.getBindingDetail(gbd);

        BindingTemplate bindingtemplate=bindingdetail.getBindingTemplate().get(0);
accesspoint=bindingtemplate.getAccessPoint().getValue();
System.out.print("fetched access point: "+accesspoint+"\n");
serviceEndpoint = accesspoint;
return accesspoint;
        //wsdlTA.append("fetched access point: "+accesspoint+"\n");

          }catch(Exception e){

                  e.printStackTrace();
                  return null;
          }
   }

public TModel getTmodel(BusinessService service){

      String tmodelKey =
service.getBindingTemplates().getBindingTemplate().get(0).getTModelInstanceDetails().getTModelInstanceInfo().g
et(0).getTModelKey();
try{
          GetTModelDetail getTmodelDetails = new GetTModelDetail();
          getTmodelDetails.setAuthInfo(rootAuthToken.getAuthInfo());
          getTmodelDetails.getTModelKey().add(tmodelKey);
          TModelDetail tmodelDetails = inquiry.getTModelDetail(getTmodelDetails);
```

```
            TModel tmodel = tmodelDetails.getTModel().get(0);
            return tmodel;
}catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }

public String getOverviewURL(TModel tmodel){

            String serviceURL = tmodel.getOverviewDoc().get(0).getOverviewURL().getValue();
System.out.println("URL"+ serviceURL);
System.out.print("Accesspoint"+ accesspoint);
            return serviceURL;

    }

public String getQoSValue(TModel tmodel){
            String QoSValue = tmodel.getCategoryBag().getKeyedReference().get(0).getKeyValue();
            return QoSValue;
    }

public String getQoSName(TModel tmodel){
             String QoSName = tmodel.getCategoryBag().getKeyedReference().get(0).getKeyName();
            return QoSName;
    }
}
```