Addis Ababa University

College of Natural Sciences

Open Source ESB Based Application Integration: Case of Ethiopian Revenue and Customs Authority

Mihret Tesfaye

A Thesis Submitted to the Department of Computer Science in Partial Fulfillment of the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

December 2016

Addis Ababa University

College of Natural Sciences

Mihret Tesfaye

Advisor: Fekade Getahun (PhD)

This is to certify that the thesis prepared by Mihret Tesfaye, titled *Open Source ESB Based Application Integration: Case of Ethiopian Revenue and Customs Authority* and submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

	Name	**Signature**	**Date**
Advisor:			
Examiner:			
Examiner:			

Abstract

Nowadays integration and interoperability becomes a key issue for organizations that work together. Enterprise Service Bus has become the ideal integration architecture for heterogeneous systems that facilitates integration between disparate applications with different hardware and software platforms.

The aim of this work is to assess services provided by Ethiopian Revenue and Customs Authority for vehicle declaration those require integration and study the workflows of the existing system. This study provided an Enterprise Service Bus product evaluation matrix and four open source ESB products evaluated and the appropriate product for implementation selected.

After discussing core ESB concepts, features and benefits, proprietary and open source ESB products are described briefly. The Enterprise Service Bus product evaluation matrix prepared by reviewing variety of research papers by different professionals and organizations and the products evaluated based on the matrix.

Based on the result from the comparison, the WSO2 ESB is used for developing the integration scenario. The development of the scenario is done using WSO2 ESB and detail information on the design and development is included.

Finally the design and implementation of the integration scenario is done using the selected ESB solution that is WSO2 ESB and the integrated system evaluated by performing functional testing. The result of the functional testing indicated a successful outcome for all the test sets.

Keywords: System integration, Enterprise Service Bus, WSO2

Acknowledgments

I would like to express my sincere gratitude to my advisor Fekade Getahun (PhD) for the continuous support of my research work, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of the project and writing of this paper. I could not have imagined having a better advisor and mentor for my research paper.

I would also like to thank the people who provide me valuable information and documents for my research requirement and who were participated in the validation and testing of the project demonstration. Without their passionate participation and input, the research work, validation and testing could not have been successfully conducted.

Table of Contents

List of Figures iii

List of Tables v

Acronyms and Abbreviations vi

Introduction 1

1.1 Statement of the Problem 2

1.2 Objective 3

1.2.1 General Objective 3

1.2.2 Specific Objective 3

1.3 Scope of the Study 4

1.4 Methodology 4

1.4.1 Data Collection Process 4

1.4.2 Review of Literature 4

1.4.3 Implementation 4

1.5 Significance 4

1.6 Structure of the Paper 5

Literature Review 6

2.1 Enterprise Application Integration 6

2.1.1 Point-to-point Architecture 6

2.1.2 Hub and Spoke Architecture 7

2.1.3 Bus Topology 9

2.2 Enterprise Service Bus 10

2.2.1 Enterprise Service Bus Functionalities 11

2.2.2 Enterprise Service Bus Products 20

2.3 Difference between EAI and ESB 34

Related Works.....35
Existing System Analysis for Declaration of Imported Vehicles.....45
4.1 Vehicle Import Permit.....45
4.2 Bank Import Permit.....46
4.3 Customs Import Declaration.....47
4.4 Imported Vehicle Declaration Scenario.....48
4.4.1 Scenario Description.....48
4.5 Imported Vehicles Declaration Use Case Diagram.....48
4.5.1 Use Cases Description.....50
ESB Design and Implementation.....55
5.1 ESB Design.....55
5.2 Implementation.....60
5.2.1 Endpoints.....61
5.2.2 Sequences.....63
5.2.3 Proxy Service.....67
5.2.4 Mediator.....69
5.2.5 Application Services.....72
5.3 Tools used for Development and Testing.....74
5.4 Testing.....75
5.4.1 Functional Testing and Result Evaluation.....79
Conclusion and Future Work.....81
Reference.....83
Annex A - Functionality Testing Document.....87
Annex B - Interview Questions.....93

List of Figures

Figure 2.1: Point-to-point integration topology....7
Figure 2.2: Hub-and-spoke integration architecture....9
Figure 2.3: Bus architecture for integration....10
Figure 2.4: Content based message routing example....12
Figure 2.5: Example on protocol transformation from JMS to File....13
Figure 2.6: Message transformation example from SOAP message to EDI message....14
Figure 2.7: Message enhancement functionality of ESB example....15
Figure 2.8: Message processing functionality in ESB....16
Figure 2.9: Process Choreography functionality of ESB.....16
Figure 2.10: Example on security functionality of ESB....18
Figure 2.11: Location transparency implementation of ESB example....19
Figure 2.12: Monitoring and management on ESB....20
Figure 2.13: WSO2 Enterprise Service Bus....22
Figure 2.14: WSO2 ESB Component Architecture....23
Figure 2.15: Overview of the functionality provided by Mule.....26
Figure 2.16: Overview of the functionality provided by Apache ServiceMix. 29
Figure 4.1: Use Case Diagram for Imported vehicle declaration....49
Figure 5.1: Imported Vehicle Declaration Process Service Integration with WSO2 ESB.....56
Figure 5.2: Business process diagram for imported vehicle declaration process....57
Figure 5.3: Import Vehicle Declaration Scenario – Summary....58
Figure 5.4: Vehicle declaration business process implemented using WSO2 ESB....59
Figure 5.5: BIPRequest Sequence....64
Figure 5.6: HSCRequest Sequence....65
Figure 5.7: TRRequest Sequence....66
Figure 5.8: PTARequest Sequence....66
Figure 5.9: IVDProxyService Proxy Service....69
Figure 5.10: Secure IVDProxyService Proxy Service from the Management Console....69
Figure 5.11: ESB test result using SoapUI1....75
Figure 5.12: ESB test result using SoapUI2....76
Figure 5.13: ESB test result using SoapUI3....76

Figure 5.14: ESB test result using TryIt1 77
Figure 5.15: ESB test result using TryIt2 78
Figure 5.16: ESB test result using TryIt3 78

List of Tables

Table 2.1: List of Open source ESBs and their Vendors and Supported Platforms 30
Table 2.2: Open Source ESBs Comparison Matrix 33

Acronyms and Abbreviations

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ASYCUDA	Automated SYstem for CUstoms Data
BPA	Business Process Analysis
BPEL	Business Process Execution Language
BPM	Business Process Management
BPR	Business Process Reengineering
BPS	Business Process Server
CAD	Cash Against Documents
CAR	Composite Application aRchives
CBE	Commercial Bank of Ethiopia
CIFS	Common Internet File System
CSV	Comma Separated Values
CXF	Celtix and XFire
EDI	Electronic Data Interchange
EE	Enterprise Edition
EIP	Enterprise Integration Patterns
ERCA	Ethiopian Revenue and Customs Authority
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
ESLSE	Ethiopian Shipping Lines & Logistics Service Enterprise
FOB	Freight On Board or Free On Board
FIX	Financial Information eXchange
FMHACA	Food, Medicine and Health Care Administration and Control Authority
FTP	File Transfer Protocol
GIS	Geographical Information System
GPL	General Public License
GUI	Graphical User Interface
HA	High Availability

HS_Code	Harmonized Code
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IMAP	Internet Message Access Protocol
InVm	intra virtual machine
IT	Information Technology
JBI	Java Business Integration
jBPM	Java Business Process Model
JDBC	Java Database Connectivity
JEE	Java Enterprise Edition
JEMS	JBoss Enterprise Middleware System
JMS	Java Message Service
JPA	Java Persistence API
JSON	JavaScript Object Notation
L/C	Letter of Credit
LGPL	Lesser General Public License
MoA	Ministry of Agriculture
MoCIT	Ministry of Communication & Information Technology
MoI	Ministry of Industry
MoT	Ministry of Trade
MoTr	Ministry of Transport
NBE	National Bank of Ethiopia
NMR	Normalized Message Router
OSGi	Open Services Gateway initiative
POX	Plain Old XML
PMS	Production Management System
POJO	Plain Old Java Object
POP	Post Office Protocol
QoS	Quality of Service
RDBMS	Relational Database Management System

SE	Standard Edition
SFTP	SSH File Transfer Protocol or Secure File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
WS	Web Service
WSDL	Web Service Definition Language
XA	eXtended Architecture
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Chapter 1

Introduction

Ethiopian Revenues and Customs Authority (ERCA) is a government organization, which is responsible for collecting revenue from customs duties and domestic taxes. In addition to raising revenue, the ERCA is responsible to protect the society from adverse effects of smuggling. It seizes and takes legal action on the people and vehicles involved in the act of smuggling while it facilitates the legitimate movement of goods and people across the border. The Authority came into existence on 14 July 2008, by the merger of the Ministry of Revenue, Ethiopian Customs Authority and The Federal Inland Revenue Authority that was formerly responsible to raise revenue for the Federal Government and to prevent contraband [7].

The Organization works in collaboration with other Governmental Agencies and Ministries: Food, Medicine and Health Care Administration and Control Authority (FMHACA), Ministry of Agriculture (MoA), Ministry of Trade (MoT), Ministry of Communication & Information Technology (MoCIT),Ministry of Industry (MoI), Ethiopian Shipping Lines & Logistics Service Enterprise (ESLSE), National Bank of Ethiopia (NBE), etc; to one or more of its business process. The current integration approach with these Authorities, Agencies or the Ministries is mostly based on paper based information exchange which is found to be costly, time consuming.

In this modern world, the business environment is changing drastically. Thus, maintaining a national competitiveness turn out to be tough. Thus, facilitating trade has become a viable and growing working option for nations to boost their shares in the global market.

Nowadays, enterprises are using multiple specialized applications to solve different aspects of their business. These different specialized applications were developed using different communication protocols and programming languages. Further, these heterogeneous applications might need to be integrated together; in order to communicate as well as to share resources and data. One of the main approaches in dealing with these challenges is the architectural style called Service Oriented Architecture (SOA).

SOA is an architectural style and a combination of methodologies that aims to achieve interoperability of remotely or locally located homogeneous and heterogeneous applications by utilizing reusable service logic. Service orientation shows variation in adopting technology for implementation, rather than focusing on the technology itself, as SOA considers the description of the problem domain before concentrating on the usage of a specific execution environment.

To exchange information between different systems or dissimilar platforms within organizations and enterprises, application integration is required. The interoperability problems get more crucial when integration is required not only between dispersed applications of one domain but require other platform dependent applications to exchange information. The information received at the receiving-end should be the same and accurate as it was sent from the sender [5]. Therefore, to integrate heterogeneous applications, a standardized communication channel is required.

Enterprise Service Bus is a middleware solution that provides high end integration between different systems [5]. It is a loosely coupled, highly distributed approach to integration. ESB provides a neutral language communication and resource sharing mechanism with a specialized unit for protocol and message translation and conversion, routing, dispatching and delivering [44]. From vendors perspective ESB can be defined as a software product which assists the developer in application integration and therefore provides the necessary infrastructure to implement routing, translation, and other integration facilities.

In this project, the integration barrier of custom clearance process in ERCA with the other involved governmental organizations will be examined and identified and an ESB-based integration solution will be devise.

1.1 Statement of the Problem

In general for import/export customs clearance process, the following problems are observed in the current working trend at ERCA.

The first problem experienced is burdensome documentation requirements. Traders are required to submit numerous documents to multiple agencies as part of the import/export clearance processes. This business process is time consuming and tiresome for traders.

The second problem observed is the use of manual clearance process. Businesses must go through a considerable number of processes to obtain clearance that can take up to 25 face-to-face interactions with around eight agencies, all of whom employ manual processes.

The basic goals of ERCA are to provide customers with equitable, efficient and quality service. ERCA works in collaboration with other Governmental Organizations and Ministry offices. In order to become more efficient, competitive and adaptive to change, the need for information system integration is a key factor. In this work, the overall gap of system integration between ERCA, Transport authority and Bank on declaring the imported vehicles process will be address.

This work focuses on system integration problem for the case of ERCA, Transport Authority and Bank for custom clearance process on imported vehicles. The work also focuses on introducing and developing an open source ESB system integration approach for the system integration between the services provided by these organizations.

1.2 Objective

1.2.1 General Objective

The general objective of this work is to design and implement an application integration using open source Enterprise Service Bus.

1.2.2 Specific Objective

In order to realize the defined general objective defined above the following specific objectives are identified:

- assess and understand the business process of ERCA,
- understand the different entities that are participating in the business process of ERCA,
- collect data about applications, data and services provided by the organizations
- Study the workflows and existing systems of the organizations,
- provide open source ESB product evaluation matrix,
- compare the different ESB solutions and select the best Open source ESB for the integration problem,

- design and implement a ESB based integration solution using the selected ESB for the selected governmental organizations,
- evaluate the integrated system using functional testing

1.3 Scope of the Study

The scope of this work is limited to studying the services and the workflow for declaration of imported vehicles process, designing and implementing of the integration scenario using the selected Open source Enterprise Service Bus solution. The organizations that participate in this business process are Ethiopian Revenues and Customs Authority (ERCA), Transport Authority and Banks.

1.4 Methodology

As a methodology for this work literature review, document analysis and interview will be used.

1.4.1 Data Collection Process

Data is gathered to make a solid foundation for the research. To identify and gather requirements a document analysis and literature review performed. For further clarification and information gathering an informal interview conducted. The interview questions used for the informal interview are attached at the back of this document on *Annex B*.

1.4.2 Review of Literature

In order to understand the state of the art in application integration, different research papers, journals and previously conducted researches reviewed.

1.4.3 Implementation

For the implementation of the demonstration, among the currently available open source ESB solutions under study the appropriate solution selected and implementation performed by the selected Enterprise Service Bus solution.

1.5 Significance

The use of the open source ESB middleware for establishment and implementation of integration aimed at improving trade competitiveness through the effective application of

trade facilitation measure and in particular testing the approach in declaring with imported vehicles business process.

The key contribution of this project is to come up with ESB based integration solution that shows the integration architecture between the different participating organizations with ERCA that work together for the common purpose of import/export procedure.

1.6 Structure of the Paper

The rest of the report is organized as follows:

Chapter two presents literature review on system integration as well as a brief history and the steps of development in this field of study which generate a foundation for the following chapters. A number of concepts related to system integration are presented briefly. The different Enterprise Service Bus products are also described in this chapter. It briefly describes the proprietary and open source ESBs. This chapter presents a comparison between proprietary and open source ESBs. It also presents the most common open source ESB products and comparison criteria. Finally it summarizes the result of comparison and selects the appropriate open source product for implementation.

The third Chapter presents related works in enterprise integration using Enterprise Service Bus and review different research papers that focus on comparing different open source ESB products.

Chapter four presents analysis of the existing system analysis under study. It describes the overall imported vehicle declaration process and all the requirements for the process.

Chapter five presents the design and implementation of the ESB based integration solution. This chapter designs the ESB based integration solution based on the system analysis done on Chapter five. The implementation of the ESB is also discussed at this chapter and the functional test result is presented.

Chapter six presents a conclusion on what has been achieved by conducting the study and further recommendations on future work in the selected area of study.

Chapter 2

Literature Review

In today's rapid growing business world, organizations are relying on technology and the need for integrating disparate applications is in high demand [6]. The notion of integration related to enterprise application integration and enterprise service bus are discussed in the next sub-sections.

2.1 Enterprise Application Integration

Enterprise application integration defined by Anurag on [9] as "*Enterprise application integration is a business need to make diverse applications in an enterprise including partner systems to communicate to each other to achieve a business objective in a seamless reliable fashion irrespective of platform and geographical location of these applications*".

Enterprise application integration can be categorized according to its design structure as Point-to-Point topology, Hub-Spoke topology and Bus topology [6].

2.1.1 Point-to-point Architecture

Point-to-point (P2P) integration is the simplest way of communication between two separate applications and it is often considered as a traditional integration approach [27]. In Point-to-point integration each application is directly connected to all the other applications it needs to communicate with. For each application to communicate in point-to-point architecture, a connector is implemented which is responsible for data transformation [6]. There exists a tight coupling between the pair of applications in Point-to-point. This means that both the applications need to be online at the same time for them to work properly. Point-to-point integration is simple since all the communication happens directly from one application to another. The communication is fast since there is no middleware that handles and processes the data before it is being received by the other application.

Point-to-point architecture can be easily build and deployed for small infrastructure. It is suitable for integrating few applications within small organizations. In this architectural model, as the number of applications increases, the connections also increase exponentially [6]. For *n* different applications to fully integrate using Point-to-point integration *n(n-1)/2*

integration points are required as shown in *Figure 2.1*. It does not scale since there is a tight dependency between the applications.

In Point-to-Point architecture, the complexity and maintenance cost increase whenever adding a new application to existing application landscape considered since it requires implementation of a new integration channel to each and every existing application and also requires modification of existing applications.

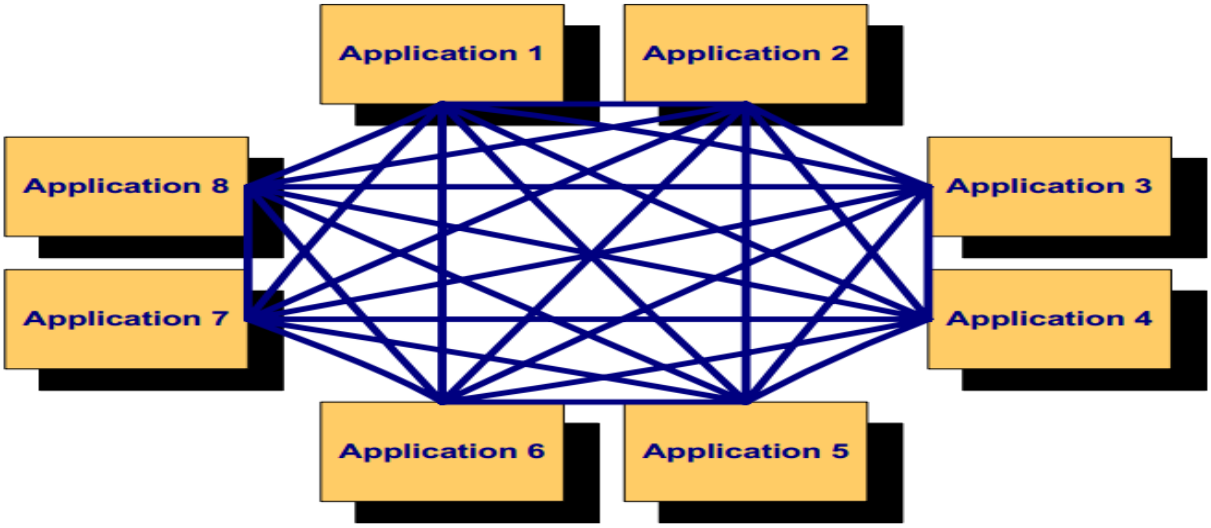

Figure 2.1: Point-to-point integration topology

In a complex enterprise computing environment, if several applications interact with each other using point to point model, the relation between applications become complex and will form a net. This will bring high maintenance costs and make the reuse of the applications difficult [40].

2.1.2 Hub and Spoke Architecture

Hub and Spoke architecture is an integration architecture that consists of a centralized broker called Hub and adapter commonly called Spoke. The Spoke is responsible for connecting the applications to the Hub. The Spoke converts application data format to the format that the Hub can understand and vice versa. The role of the Hub in Hub and Spoke architecture is brokering all messages and taking care of content transformation/translation of the incoming message into the format the destination system can understand. It also handles the routing of incoming messages to their right destination application(s) [6, 9].

Unlike Point-to-point architecture, where each application has many integration points for each application it needed to integrate, with the hub and spoke architecture, each application has only one integration point which is connected to the broker. This simplifies the integration process greatly when the amount of integrations needed for different application grow. With hub and spoke pattern a single integration point per application is needed [27]. Hub and Spoke architecture has multiple advantages over Point-to-point. It enables loose coupling since it uses asynchronous communication where recipients do not need to be aware of each other or even know whether the other application is currently online. The hub will receive messages from the sending application and route them to the receiving application(s). If one or all of the receivers are unavailable at that time, then it can save the message internally until such time that all the receiving applications have successfully received the message [27].

For hub and spoke architecture having a single hub makes the system management easy since all the configuration is done in one central location making it easier to modify existing integration and create new ones. But at the same time makes scalability difficult. At some point of time as number of messages increase, scalability gets dependent on hardware. With the Hub and Spoke architecture the problem is that it creates a single point of failure. If the platform that supports the hub becomes unavailable or overloaded, it can potentially generate a single point of failure.

The diagram on *Figure 2.2* shows the hub and spoke integration architecture.

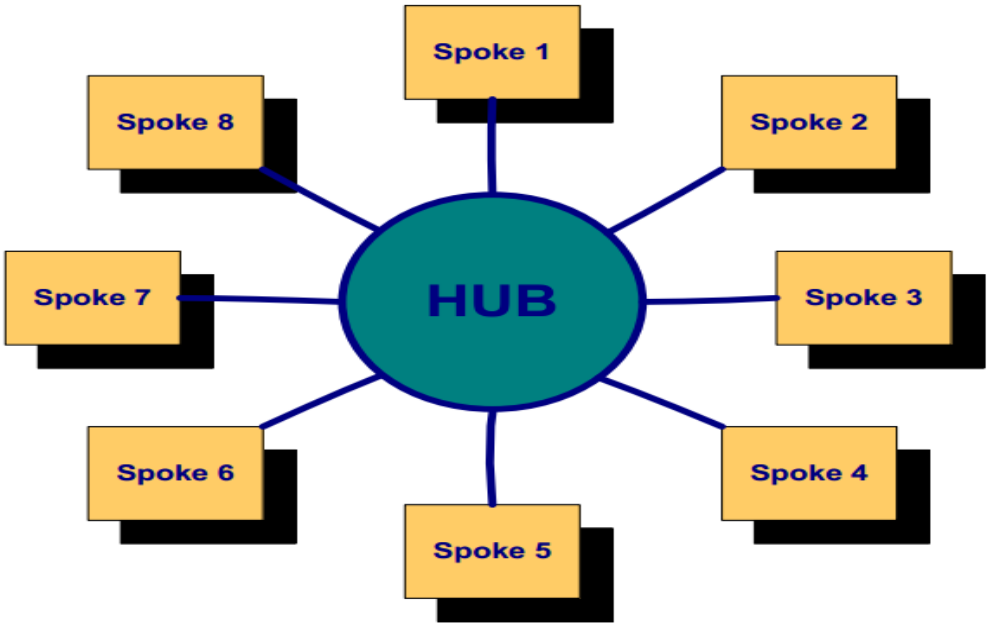

Figure 2.2: Hub-and-spoke integration architecture

2.1.3 Bus Topology

The Hub and Spoke architecture lack standards and the centralized broker model is the single point of failure. And the problem of single component leads to a single point of the entire network for this architecture. To overcome the problems of Hub and Spoke architecture, as a solution the bus model emerged.

In bus architecture, a centralized routing component is used. In this architecture, the components can be grouped and hosted in different places in the network. For scalability purpose, it can also be duplicated geographically. As the bus model evolved, additional functionalities like security, transaction processing and error handling included.

As described in [6], the bus model is a lightweight, suitable and reliable integration solution which is abstracted solution with consistent pattern that can be designed with less coding and with no modification to the applications. The model finally came to be known as Enterprise Service Bus (ESB) [6]. *Figure 2.3* below shows the bus architecture for integration.

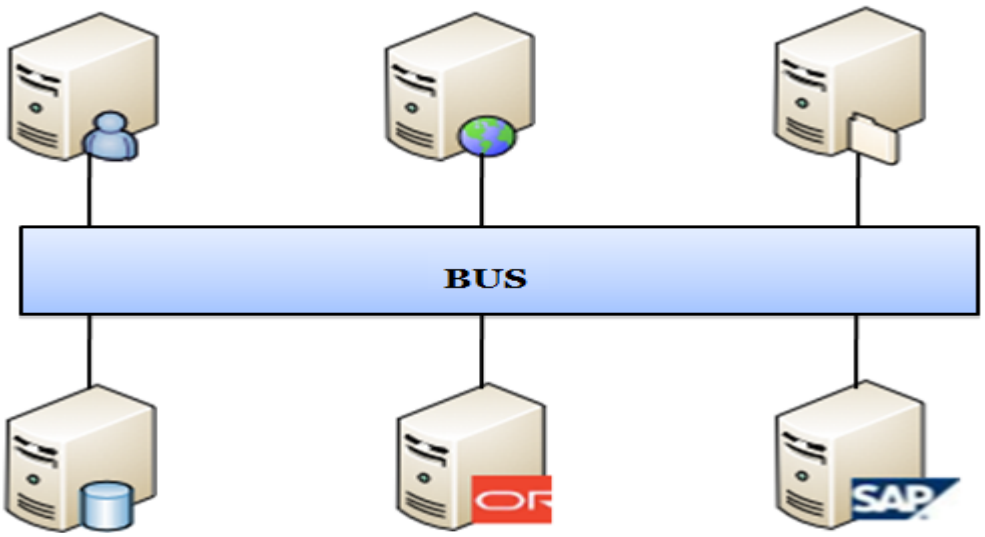

Figure 2.3: Bus architecture for integration

2.2 Enterprise Service Bus

An Enterprise Service Bus is a software architecture construct that enables communication among various applications. Instead of letting each of the heterogeneous applications communicate directly with each other, each application simply communicates with the ESB, which provides transforming and routing the messages to their appropriate destinations [38]. This decouples systems from each other, allowing them to communicate without dependency on or knowledge of other systems on the bus.

An enterprise service bus (ESB) is a middleware solution that enables interoperability among heterogeneous environments using a service-oriented model [13]. It is a standardized, message based, distributed integration infrastructure that provides routing, invocation and mediation services to facilitate the interactions of disparate distributed applications and services in a secure and reliable manner [14]. The general goal is then to provide messaging and integration without writing code by only configuring. Therefore generic components are provided which can be configured to realize a desired scenario.

Applications (and integration components) in the ESB are abstractly decoupled from each other, and connect together through the bus as logical endpoints that are exposed as event-driven services. With its distributed deployment infrastructure, an ESB can efficiently

provide central configuration, deployment, and management of services, which are distributed across the extended enterprise [1].

One advantage of connecting clients and services via an enterprise service bus is that clients need only look for services in a single location: The enterprise service bus. If a service is moved from one server to another, it is only needed to reconfigure the end point of the ESB. The clients still just access the service via the ESB.

Adding a new application is also simpler than before. New application can be connected to the ESB with the transport protocol and technology adapter suited for the application. The integration flows that connect the new application with the existing applications can be handled within the ESB.

An ESB from an integration vendor perspective is a product offering that provides integration functionality, a developer toolset, and a management environment [12].

2.2.1 Enterprise Service Bus Functionalities

ESB provides various functionalities to facilitate the integration between disparate applications with different hardware and software platforms. According to [12, 19, 45], ESB usually has more or less the following functionality.

2.2.1.1 Routing

Routing is the ability to channel a request to a particular service provider on deterministic or variable routing criteria. Determining the ultimate destination of an incoming message is an important functionality of an ESB that is categorized as message routing.

Multiple applications are involved in most integration projects. These applications can be the target application for a particular incoming message. The ESB has to determine to which service provider(s) a message must be sent, based on different rules and logic. The core functionality of ESB that handles this kind of responsibility is known as message routing. There are different types of routing: static or deterministic routing, content based routing, policy-based routing, or complex rules-based routing [45].

In static or deterministic routing, the routing of traffic between given pair of endpoints is preprogrammed. The message is routed to a pre-defined endpoint address that is hard-wired in the module.

In content-based routing, messages routed to their ultimate destination based on their content. The Content-based routing provides the ability to route messages to specific addresses based on information contained in the message headers and payloads. The *Figure 2.4* below is taken from [12] and it shows an example of message routing based on the content of an incoming message.

Policy-based routing routes the messages based on policies rather than by an explicitly specified destination. By using policy-based routing, policies can be implemented to selectively cause messages to be routed through different paths. Policy Based Routing provides a flexible mechanism for the ESB to customize the operation of routing incoming messages to the appropriate endpoint based on policies. Rule based routing routes the message based on complex rules routing criteria.

In the example illustrated in the figure below, *Figure 2.4* [12], the insurance request consists of an element named *requesttype* that specifies the kind of insurance request applied for by the customer using the Internet application. Based on the value of this element, the message is routed to the travel or the car insurance service [12].

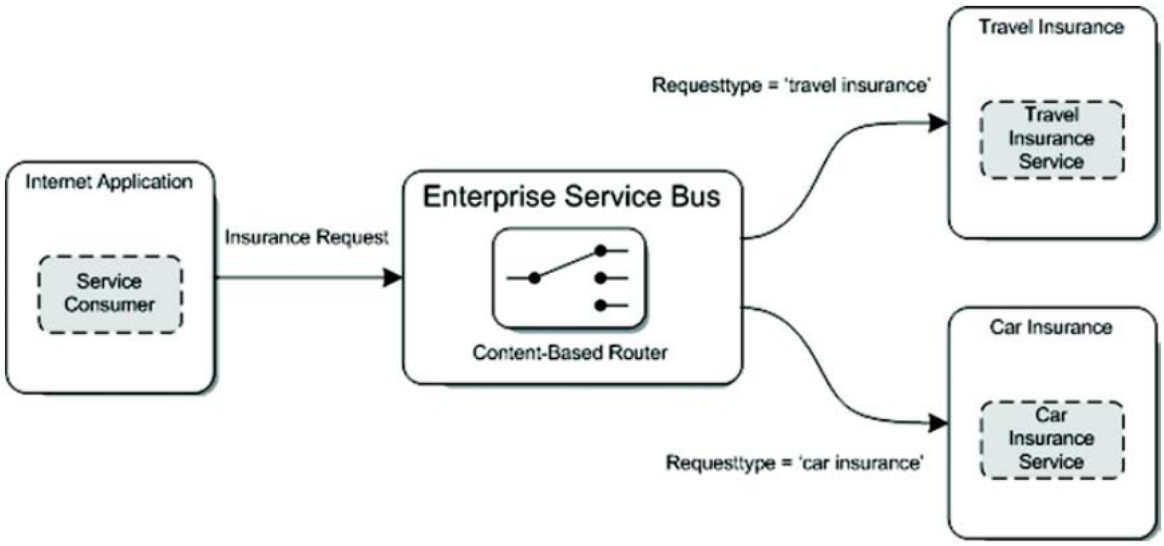

Figure 2.4: Content based message routing example

2.2.1.2 Protocol Transformation

Protocol transformation is the ability to accept one type of protocol from the consumer as input and then communicate to the service provider through a different protocol. An ESB

should be able to seamlessly integrate applications with different transport protocols like HTTP(S) to JMS, FTP to a file batch, or SMTP to TCP. The component in an ESB offering transport protocol conversion is typically referred to as Protocol Adapters.

In the example illustrated on the figure below, a service consumer uses a different transport protocol than the service provider. The service consumer communicates through JMS and the service provider is a legacy system that is only capable of importing and exporting files in a batch. The figure on *Figure 2.5*[12] shows the transport protocol conversion of the example discussed that is protocol conversion from JMS to File.

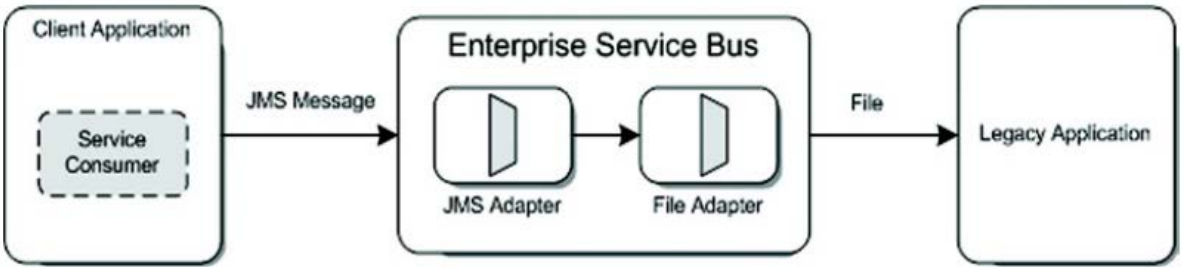

Figure 2.5: Example on protocol transformation from JMS to File

In the example shown in *Figure 2.5*, a client application sends a JMS message to the ESB. A JMS adapter accepts the JMS message and forwards it to the file adapter, which writes the content of the JMS message to the file system of a legacy application.

2.2.1.3 Message Transformation

Message transformation is the ability to convert the structure and the format of the incoming business service requests to the structure and format expected by the service provider. The ESB provides functionality to transform message from one format to the other based on open standards like XSLT and XPath.

Implementing integration between a service consumer and a service provider often requires a transformation of the message format. In the example shown in *Figure 2.6*, the content of the JMS message cannot be forwarded as is to the legacy application. There is a need for logic that transforms the message format to the expected format of the service provider. The ESB core functionality that helps with changing the message format is known as the message transformation functionality.

The example used in *Figure 2.6* [12] shows a transformation from a SOAP message to an electronic data interchange (EDI) message.

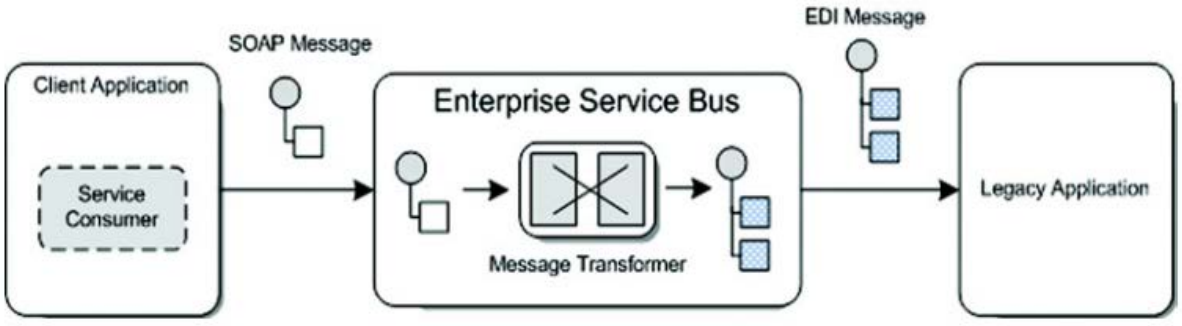

Figure 2.6: Message transformation example from SOAP message to EDI message

2.2.1.4 Message Enhancement

Message enhancement is the ability to add or modify the information contained in the message as required by the service provider. An ESB should provide functionality to add missing information based on the data in the incoming message by using message enhancement.

The message transformation process can change a source message format to a target message format. But to be able to create the correct outgoing message that will be sent to the target application, it might be necessary to add additional data or convert the existing data. A common way to add data to a message is by retrieving it from a database based on certain element values of the incoming message. An example of such an element is a client identifier. The destination of the incoming message with the client identifier can be an application that requires some extra client information that is not available in the incoming message. The ESB can then retrieve this information from a database based on the client identifier in the incoming message.

The functionality described here can be categorized as a message enhancement capability and is closely related to message transformation. The main difference between these functionalities is that message transformation deals with data that's already available in the incoming message, and message enhancement deals with data that must be retrieved from a (external) data source, for example a database [12].

The figure below, *Figure 2.7* [12], illustrate message enhancement in CRM [12] in order to retrieve detail using customer identifier.

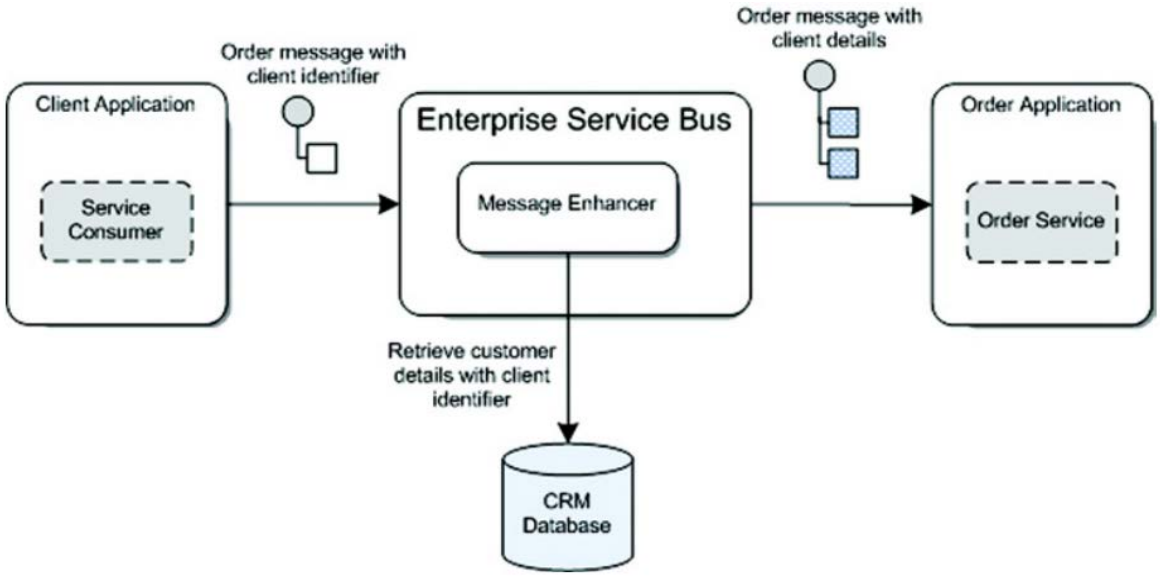

Figure 2.7: Message enhancement functionality of ESB example

In general, types of message enhancement could be data format conversion, supplement data not included in original message, data conversion i.e. spaces to zero or rule based enhancement.

2.2.1.5 Service Mapping

Service mapping is the ability to translate a business service into a corresponding service implementation and provide binding and location information. It could be implemented through XML, a database, or embedded within the mediator ESB component. Usually service mapping contains core information: implementation service name, service protocol and binding information, protocol specific information and service-specific routing information.

2.2.1.6 Message Processing

Message processing is the ability to manage state and perform request management by accepting an input request and ensuring delivery back to the client via message synchronization as shown in *Figure 2.8*.

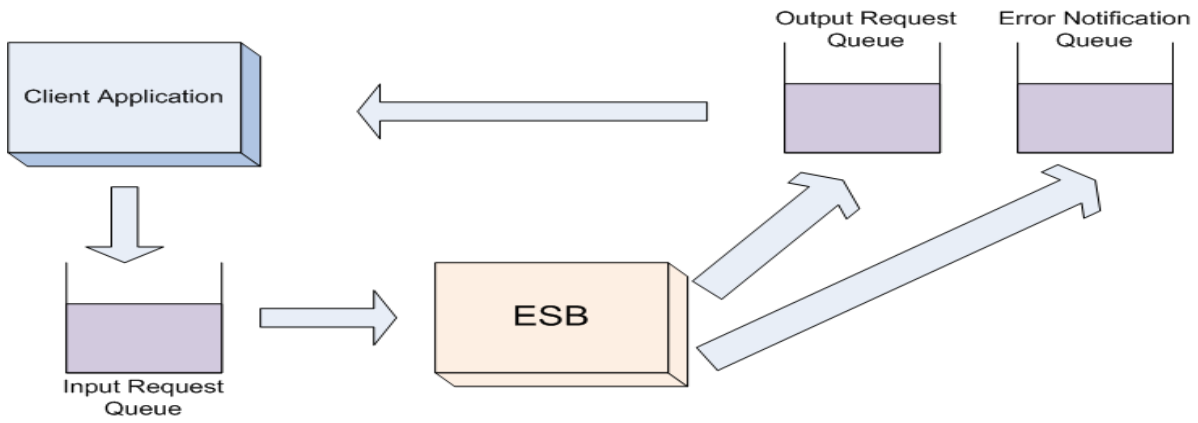

Figure 2.8: Message processing functionality in ESB

2.2.1.7 Process Choreography

Process choreography is the ability to manage complex business processes that require the coordination of multiple business services to fulfill a single business service request. Usually it is BPEL based. Process Choreography can be thought of as a manifestation of a use case or business process.

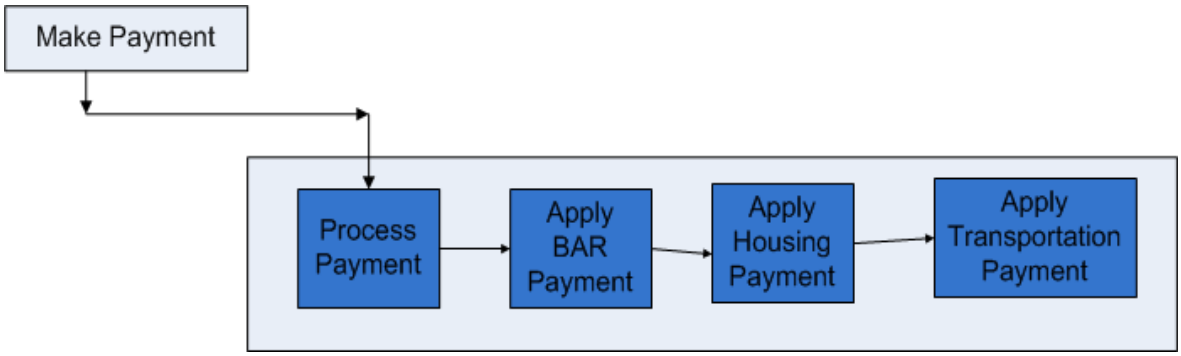

Figure 2.9: Process Choreography functionality of ESB.

Each of the business nodes on the above diagram, *Figure 2.9*, can be an independent business service.

2.2.1.8 Service Orchestration

Service Orchestration is the ability to manage the coordination of multiple implementation services. It can be BPEL based but it usually implemented through inter-service communication or aggregate services. The difference between Service Orchestration and Process Choreography is based on the type of service being coordinated. In the case of

Process Choreography, the service coordinated is business services and in case of Service Orchestration, implementation services are coordinated.

2.2.1.9 Transaction Management

Transaction management is the ability to provide a single unit of work for a business service request by providing a framework for the coordination of multiple resources across multiple disparate services. Specifically, the ESB should provide a compensatory transactional framework for a service request.

2.2.1.10 Security

Security is the ability to protect enterprise services from unauthorized access. Authentication, authorization, and encryption functionality should be provided by an ESB for securing incoming messages to prevent malicious use of the ESB as well as securing out-going messages to satisfy the security requirements of the service provider.

Because ESBs often deal with business-critical integration logic that involves a substantial number of applications, an ESB must provide ways to authenticate and authorize incoming messages.

For messages that might be intercepted for malicious purposes, encryption is an important feature that an ESB must be able to provide.

The example in *Figure 2.10* [12] illustrates how the authentication inside an ESB can be implemented. Besides authentication, authorization can also be configured for an integration flow. By using authorization, the functionality of a service provider can be secured on a method level so that, for example, a group of users can be granted different access than an administrator user. The example below also implements encryption for the outgoing message before it is sent to the service provider. This is another part of the security functionality an ESB should be able to implement. Service providers can have all kinds of security measures implemented, and an ESB should be able to construct an outgoing message that has the right security values set. For example, to ensure that a message cannot be read by other parties, a message can be encrypted with the public key of the service provider in the ESB [12].

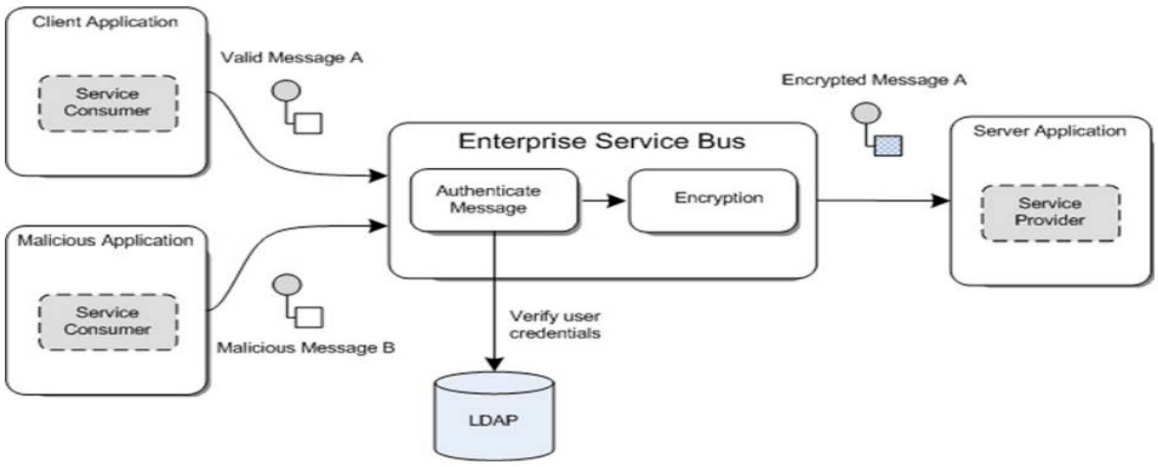

Figure 2.10: Example on security functionality of ESB

Security involves the confidentiality, integrity and availability of messages sent over the ESB. This example shows an implementation of confidentiality via an authentication mechanism.

2.2.1.11 Location Transparency

When a service consumer communicates with a service provider via the ESB, the consumer doesn't need to know the actual location of the service provider. This means that the service consumer is decoupled from the service provider and that a service provider's new server location has no impact on the service consumer. The core functionality of an ESB that provides this capability is known as location transparency.

The ESB allows decoupling the service consumer from the service provider location. The ESB provides a central platform to communicate with any application necessary without coupling the message sender to the message receiver.

It is possible to implement the location transparency within the ESB with a simple XML configuration, a database, or a service registry. The approach depends on the requirements, such as dynamic configuration capabilities and the need for additional information about service providers (e.g., quality of service). The simplest and common implementation of location transparency is the configuration of service provider endpoints in a static XML file. Dynamic configuration can be implemented with a hot-deployment model for location configuration files or with locations stored in a database.

Figure 2.11 [12] shows a graphical overview of the options that are available when implementing location transparency with an ESB.

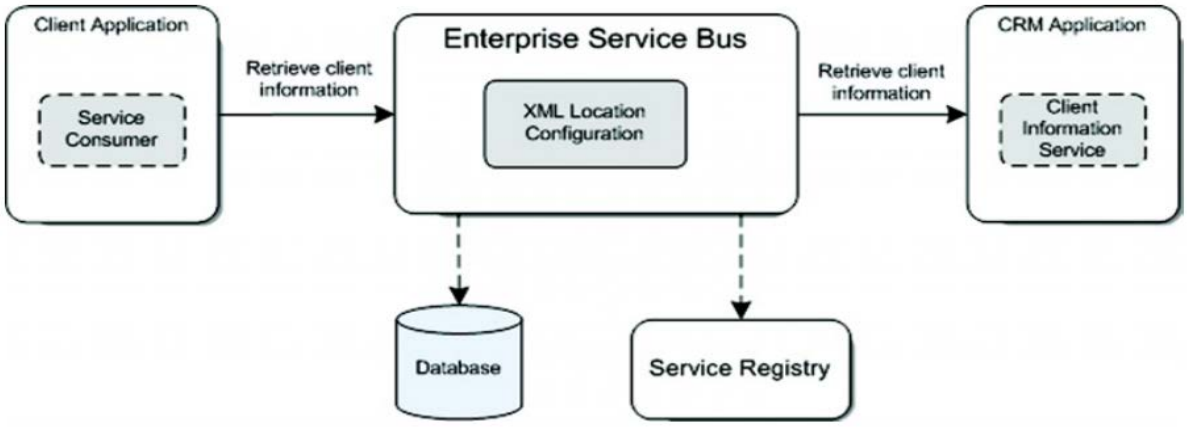

Figure 2.11: Location transparency implementation of ESB example

The *Figure 2.11* depicts a simple case in which an application needs client information from a CRM application. Because an ESB is used, the location of the client information service within the CRM application is transparent to the service consumer. Notice that when the location of the client information service changes, only the location configuration within the ESB has to be updated.

2.2.1.12 Monitoring and Management

Managing and monitoring an ESB environment becomes complex because of the large set of capabilities an ESB provides. Therefore, the management and monitoring functionality consists of multiple parts, and each is responsible for a component of the ESB. For the messaging layer in the ESB, the management and monitoring environment will, for instance, involve managing the queues and monitoring the message size and message throughput of queues. For web services provided by the ESB, monitoring will involve such things as whether the web service is up and running and how many calls are made per minute; management will address the number of instances that are running for a web service.

The graphical representation of monitoring and management functionality of an ESB is shown in the *Figure 2.12* [12] below.

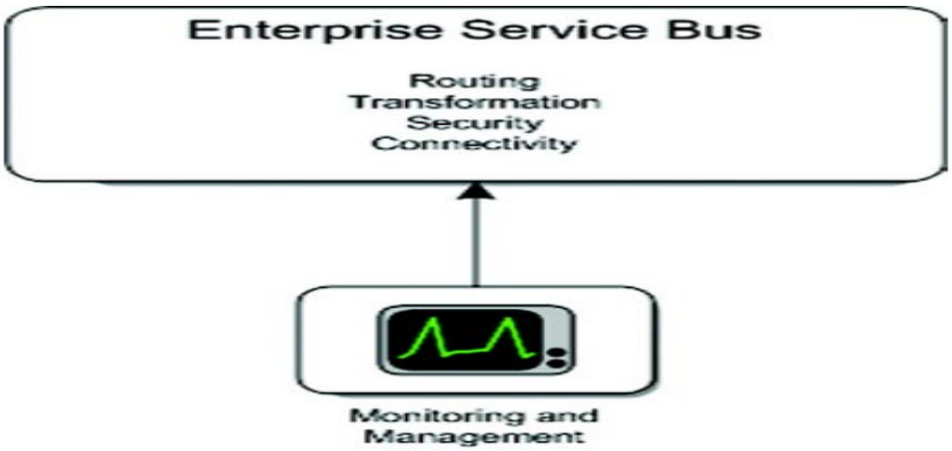

Figure 2.12: Monitoring and management on ESB

The ESB is a central product within the environment and therefore monitoring and management capabilities are vital.

2.2.2 Enterprise Service Bus Products

Enterprise Service Bus is a software architecture that provides capabilities including connectivity, communications, message queuing, message routing, message transformation, reliable messaging, protocol transformation and binding, event handling, fault handling, message level security, and so on. It enables loose coupling between service consumers and service providers, and thus enhances the ability to rapidly change and introduce new Business Services into the environment [27].

In general Enterprise Service Bus products can be categorized in to two as proprietary and open source Enterprise Service Bus products. In this chapter the different enterprise service bus products will be examined, basic information about the products presented and a product will be selected using a well-defined criteria.

2.2.2.1 Proprietary Enterprise Service Bus Products

Proprietary software is applications that are entirely written, updated and maintained by their owners. The organization that owns the software holds the exclusive rights to the source code, meaning that users that want to use the software need to acquire a software license, which entitles usages according to specific terms. So even though it is paid for a license, altering the software to own desires not allowed. A clear disadvantage of proprietary

software is the limited access to the source code and the other constraints that may have been set forth by the terms and conditions in the license agreement.

The term closed source ESB refers to ESB products that have a usage-based license fee and for which the source code is not freely available. The proprietary ESB are significantly less flexible when compared to open source ESBs, since the development work is handled almost exclusively via graphical user interfaces in the products, no direct access to the source code is possible, their platform is "closed" and not based on open standards.

Some of commercial ESB Products are Websphere ESB[1] from IBM, ActiveMatrix ESB[2] from Tibco, Oracle ESB[3] from Oracle and BizTalk Server[4] from Microsoft.

2.2.2.2 Open Source Enterprise Service Bus Products

Open source software application is entirely written, updated and maintained by a community, allowing everyone to view the code. This means developers and organizations are able to learn from the code, contribute to it, and reuse it in their own projects. Open-source software also comes with a license, but does not restrict the freedom of the end-users the way proprietary licenses do.

Open source ESBs do have a license like the Apache or GPL license, but don't have a usage-based license fee and the source code is freely available. In addition to the free source code, services and support can be provided for free for open source ESBs.

In the market, there are quite a lot of open source ESBs. In this work the focus was on products which attracted more attention in research papers and mentioned in different published journal articles as a possible options for integration solution. The list of the open source ESBs and their description is presented in the subsections below.

2.2.2.2.1 WSO2 ESB

WSO2 is a company started in 2005 and released their first Enterprise service Bus in 2007 and latest version is 4.9.0. It is built on Apache Synapse ESB. It uses ActiveMQ for message

[1] http://www-01.ibm.com/software/integration/wsesb/library/
[2] http://www.tibco.com/products/automation/application-integration/activematrix-businessworks/enterprise-service-bus
[3] http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html
[4] https://msdn.microsoft.com/en-us/biztalk/biztalk-esbtoolkit.aspx

queuing and Axis2 as web services. It is concentrated on using web services as message procession components. The WSO2 Enterprise Service Bus is a fast, lightweight, and user-friendly ESB distributed under the Apache Software License v2.0. WSO2 ESB allows system administrators and developers to conveniently configure message routing, mediation, transformation, logging, task scheduling, failover routing, load balancing, and more. It also supports transport switching, eventing, rule-based mediation, and priority-based mediation for advanced integration requirements. The ESB runtime is designed to be completely asynchronous, non-blocking, and streaming based on the Apache Synapse mediation engine [38].

Using WSO2 ESB a variety of enterprise integration patterns (EIPs) can be performed, including filtering, transforming, and routing SOAP, BINARY, plain XML, and text messages that pass through business systems by HTTP, HTTPS, JMS, mail, etc.

WSO2 ESB is a full-fledged, enterprise-ready ESB. It is built on the Apache Synapse project, which is built using the Apache Axis2 project. All the components are built as OSGi bundles [38].

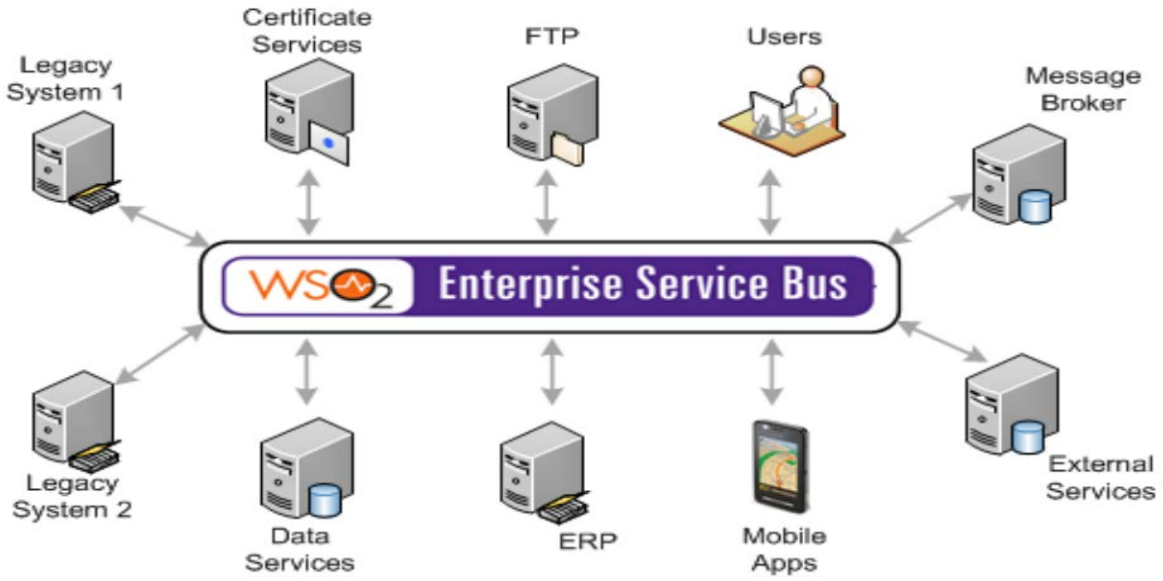

Figure 2.13: WSO2 Enterprise Service Bus

The component architecture of WSO2 ESB is illustrated in the *Figure 2.14* [38].

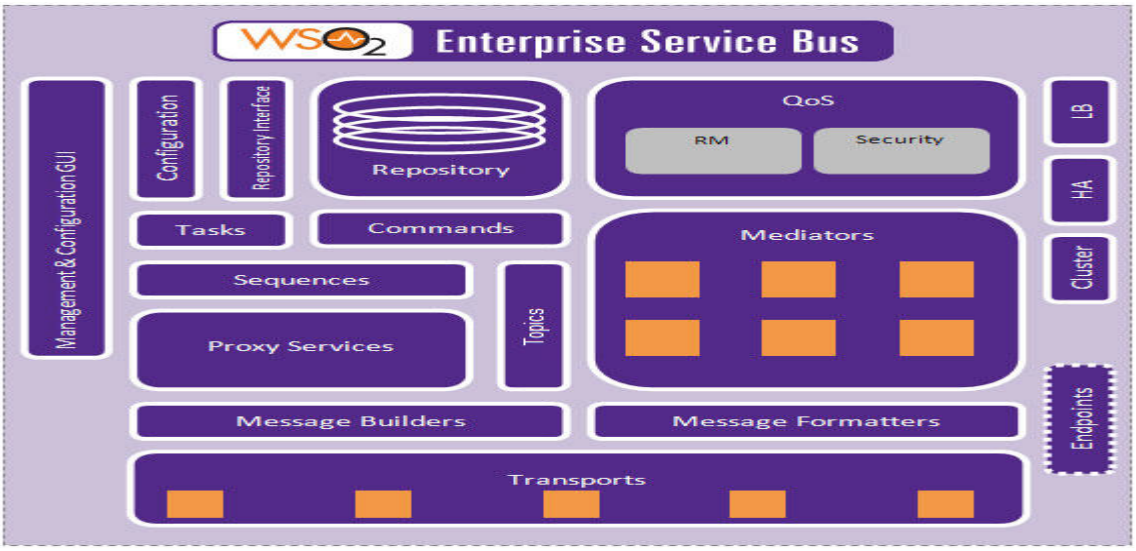

Figure 2.14: WSO2 ESB Component Architecture

WSO2 ESB provides transports, endpoints, proxy service, API, Sequencing, Mediation, QoS, registry services as is briefly described here under.

A transport component in WSO2 ESB is responsible for carrying messages that are in a specific format. WSO2 ESB supports all the widely used transports including HTTP/s, JMS, and VFS, and Domain-specific transports like FIX. An external destination (such as a service) for a message defined using endpoints component. An endpoint can be specified as an address endpoint, WSDL endpoint, a load balancing endpoint, and more. Proxy services in WSO2 ESB are virtual services that receive messages and optionally process them before forwarding them to a service at a given endpoint. This component allows performing necessary transformations and introducing additional functionality without changing the existing service. API components are anchored at a user-defined URL context, much like how a web application deployed in a servlet container and the API will only process requests that fall under its URL context. A topic component allows services to receive messages when a specific type of event occurs by subscribing to messages that have been published to a specific topic. Mediators are individual processing units that perform a specific function, such as sending or filtering messages. WSO2 ESB includes a comprehensive mediator library that provides functionality for implementing widely used enterprise integration patterns. Sequences are the configuration component for mediators. Sequences in WSO2 ESB allow organizing mediators to implement pipes and filter patterns. The Quality of

Service (QoS) components implement security for the proxy services. WSO2 ESB also provides a registry with a built-in repository that stores the configuration and configuration metadata that define messaging architecture. The Management Console of WSO2 ESB provides a graphical user interface that allows to easily configuring the components mentioned above as well as managing and monitoring the ESB. It is also possible to easily write a custom component to provide additional functionality using various technologies and plug-in to the ESB.

WSO2 ESB has various features and capabilities. It can connect anything to anything by supporting a number of transports, multiple formats and protocols. It has limitless routing, mediation and transformation. It has a complete control over security including authentication, authorization and entitlement. In order to have a high performance and high availability solution the ESB supports thousands of concurrent non blocking HTTP(s) connections per server that will allow even the most demanding organizations to utilize the ESB. The WSO2 ESB comes with comprehensive monitoring capabilities that include: System status, System statistics with graphs, Mediation statistics with graphs, Mediation tracer, SOAP tracer Logs configuring and monitoring. It is also possible to shutdown and restart the ESB, gracefully and forcefully through the management console and the graphical user interface. It has built-in capabilities such as: Content based routing, Service virtualization, Load balancing, Fail-over sending, Protocol switching, Message transformation, Logging & monitoring, Message splitting and aggregation, Enterprise integration patterns, Request throttling and Response caching. WSO2 ESB provides a set of management services and a graphical user interface to configure/manage/monitor the running ESB server. Features of this graphical console includes: Sequence editor, Proxy Service editor, Endpoints/Local Entry editor, Task scheduler, Built-in registry browser, policy editor, Predefined security scenarios, User stores, Keystores, Configure data sources, Transport management, Try-It for services, and Logs, trace and statistics monitor. It has also built-in support for reading from or writing to Databases.

2.2.2.2.2 Mule ESB Community

Mule Enterprise Service Bus Community Edition is an integration tool developed by MuleSoft. It is a lightweight event-driven enterprise service bus and an integration platform

[3]. It is Java-based enterprise service bus and integration platform that allows developers to connect applications together quickly and easily, enabling them to exchange data. Mule ESB enables easy integration of existing systems, regardless of the different technologies that the applications use, including JMS, Web Services, JDBC, HTTP, and more[5].

Mule ESB includes powerful capabilities that include:

- **Service creation and hosting** — expose and host reusable services, using Mule ESB as a lightweight service container
- **Service mediation** — shield services from message formats and protocols, separate business logic from messaging, and enable location-independent service calls
- **Message routing** — route, filter, aggregate, and re-sequence messages based on content and rules
- **Data transformation** — exchange data across varying formats and transport protocols

Mule ESB provides a very solid engine for mediation, routing, and lightweight orchestration. Mule ESB has two versions; Mule ESB community and Mule ESB enterprise. The Mule ESB enterprise is only available for subscribers [5].

Mule ESB provides different functionalities. The basic functionalities are:

- Composing several existing fine-grained components into a single higher order composite service. This can be done to achieve appropriate "granularity" of services and promote reuse and manageability of the underlying components.
- Transformation between canonical data formats and specific data formats required by each application or service being orchestrated. For example, transforming between CSV, Cobol copybook or EDI formats to either SOAP/XML or JSON.
- Transport protocol negotiation between multiple formats (such as HTTP, JMS, and JDBC). Mule treats databases like another "service" by making JDBC just another transport (or endpoint) where data can be accessed.

[5] https://www.mulesoft.com/resources/esb/what-mule-esb

- Providing support for multiple interfaces for the purpose of supporting multiple versions of a service for backwards compatibility or alternatively and to allow for multiple channels to the same underlying component implementation.
- For application orchestration, it provides consistency around the way security and monitoring policies are applied and implemented. Scalability and availability can be achieved by using HA clustering.

The overview of the functionalities provided by Mule is demonstrated in *Figure 2.15* [12] below.

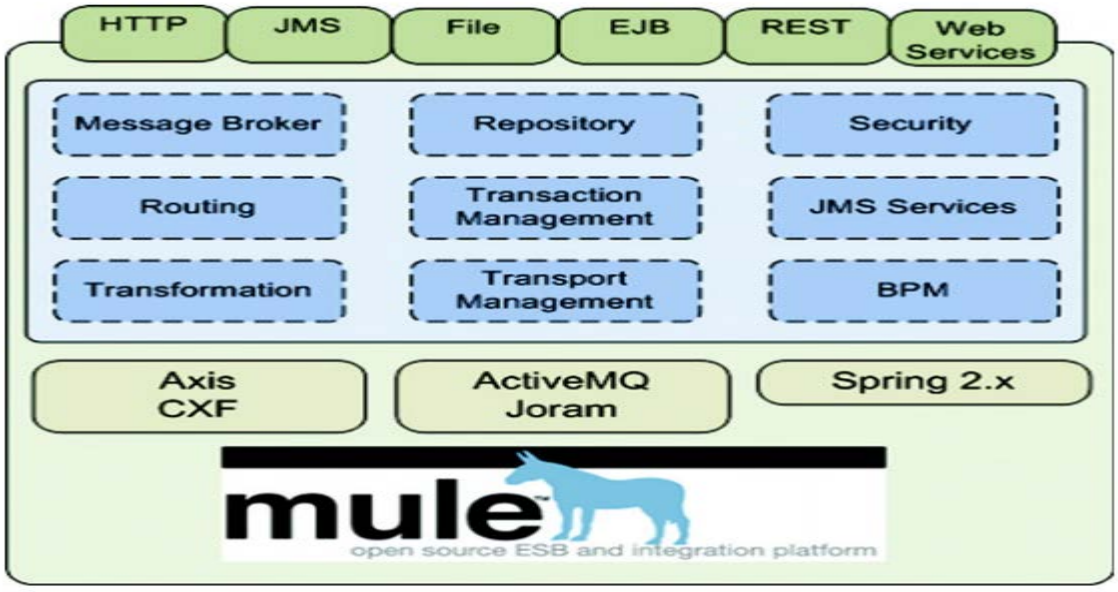

Figure 2.15: Overview of the functionality provided by Mule.

The figure shows some examples of open source frameworks that can be integrated with Mule, including CXF and ActiveMQ.

2.2.2.2.3 JBoss ESB

The JBoss Enterprise SOA Platform is a framework for developing enterprise application integration and service-oriented architecture solutions. It is made up of an enterprise service bus (JBoss ESB) and some business process automation infrastructure. It allows building, deploying, integrating and orchestrating business services [15].

The JBoss Enterprise SOA Platform provides a comprehensive server for data integration needs. On a basic level, it is capable of updating business rules and routing messages through an Enterprise Service Bus [15].

The JBoss Enterprise SOA Platform's JBossESB component which is originated in 2006, supports multiple transports and protocols, listener-action model so that services can loosely-couple together, Content-based routing through the JBoss Rules engine, XPath, Regex and Smooks, integration with the JBoss Business Process Manager (jBPM) in order to provide service orchestration functionality, integration with JBoss Rules in order to provide business rules development functionality and integration with a BPEL engine.

Furthermore, the ESB allows integrating legacy systems in new deployments and have them communicate either synchronously or asynchronously. In addition, the enterprise service bus provides an infrastructure and set of tools that can be configured to work with a wide variety of transport mechanisms (such as e-mail and JMS), be used as a general-purpose object repository, allow implementing pluggable data transformation mechanisms, support logging of interactions.

JBoss ESB must be deployed on the JBoss Java EE web server, and offers tight interoperability with other JBoss technologies such as JEMS and JBossMQ. Because the JBoss ESB is designed to operate in conjunction with other JBoss products, its release schedule tends to be tied to these products. The JBoss ESB roadmap is often less aggressive than that of other integration products.

Basic Features of JBoss are[6]:

- Support for general notification framework. Transports supported include JMS, InVm, TCP/IP, email, database or file system. JMS and SQL transaction integration.
- More seamless integration when deployed into JBossAS.
- jBPM integration.
- Web Services support (jbossws-native and jbossws-cxf).
- Improved deployment and configuration, using a specific ESB server.

[6] http://jbossesb.jboss.org/features.html

- Support for data transformations using Smooks 1.2.5 or XSLT.
- Listeners and action model to support loose-coupling of interaction steps.
- Content based routing using Drools, regexp or XPath.
- Gateways to allow non-ESB aware traffic to flow into the ESB.
- Graphical configuration editor.
- High performance and reliability

2.2.2.2.4 Apache ServiceMix

Apache ServiceMix is an Enterprise Service Bus licensed under Apache license V2. ServiceMix Project started in 2005 and it is now in version 6.0.0. It combines the functionality of a Service Oriented Architecture and the modularity. The adoption of a Service Bus allows decoupling the applications together and reducing dependencies. Messages are used to wire the applications and connectors to exchange information using different protocols or communications mode like FTP, HTTP, WebServices and so on [31].

Apache ServieMix is a flexible, open-source integration container that unifies the features and functionality of Apache ActiveMQ, Camel, CXF, and Karaf into a powerful runtime platform. It provides a complete, enterprise ready ESB exclusively powered by OSGi. There is a commercial Enterprise Service Bus implementation called Fuse ESB that is based on ServiceMix and can be thought of as the commercial version of ServiceMix [27].

ServiceMix is lightweight and easily embeddable, has integrated Spring support and can be run at the edge of the network (inside a client or server), as a standalone ESB provider or as a service within another ESB. You can use ServiceMix in Java SE or Java EE application server [31].

It is a distributed ESB built from the ground up on the Java Business Integration specification JSR 208. The goal of JBI is to allow components and services to be integrated in a vendor independent way, allowing users and vendors to plug and play.

ServiceMix uses ActiveMQ to provide remoting, clustering, reliability and distributed failover. It can be embedded into a JEE application server such as JBoss, Oracle Weblogic or IBM Websphere.

ServiceMix includes a complete JBI container supporting all parts of the JBI specification including NMR, JBI Management MBeans and full support for the JBI deployment units with hot-deployment of JBI components. It also provides a simple to use client API for working with JBI components and services. ServiceMix includes many JBI components including HTTP, JMX, CXF, BPEL, etc [31].

The main features of Apache ServiceMix are reliable messaging with Apache ActiveMQ, messaging, routing and supporting Enterprise Integration Patterns and RESTful web services. Through additional installable features, ServiceMix also supports BPM engine, full JPA, XA transaction management, and legacy support for the JBI standard. The various functionalities provided by Apache ServiceMix illustrated in the diagram *Figure 2.16* below.

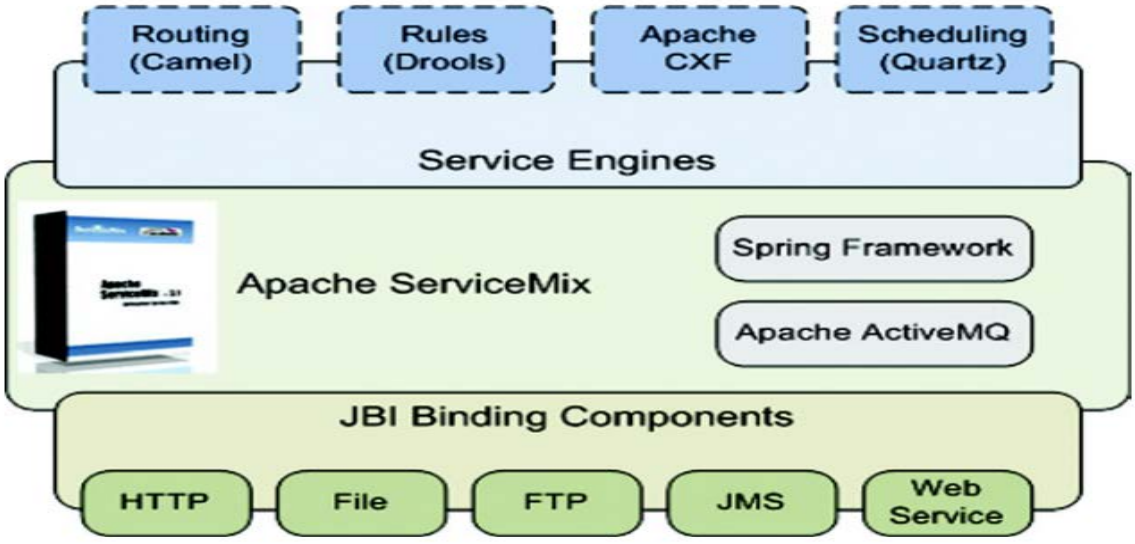

Figure 2.16: Overview of the functionality provided by Apache ServiceMix.

2.2.2.3 Comparison of the Open Source Enterprise Service Buses

There is no doubt on the fact that an Enterprise Service Bus (ESB) is important for company to integrate systems at hand and for the ones to be implemented in the future. But the biggest question and the biggest work is in which Enterprise Service Bus product to choose. There are multiple vendors that provide an open source Enterprise Service Bus with variety of functionalities or variety of enterprise integration patterns. For this work the research result of other related works by professionals are referred and their evaluated results are

considered for deciding the appropriate open source product to use. The assessment was made using publically available product documentations, published documents, and whitepapers.

The considered open source Enterprise Service Bus products for selection as described on Section 4.2 of this document are; WSO2 ESB ,Mule ESB, JBoss ESB and ServiceMix. The open source products their vendors, license information and the supported plat form information is summarized on *Table 2.1*.

Table 2.1: List of Open source ESBs and their Vendors and Supported Platforms

Open Source ESB	**Vendor**	**Software License**	**Platform Support**
JBossESB	Red Hat	LGPL v2.1	Windows/Linux
Mule ESB	MuleSoft	CPAL	Windows /Ubuntu/Red Hat /Oracle Solaris /Apple Mac/IBM
Apache ServiceMix	Apache	Apache Software License v2.0	Windows/Linux/ Unix/MacOS
WSO2 ESB	WSO2	Apache Software License v2.0	Windows/Linux/Solaris

2.2.2.4 Open Source ESBs Comparison Matrix

After considering and reviewing the research reports by different companies and expert professional, the comparison matrix for the open sources under study is prepared. The comparison criteria considered are described in the following paragraphs.

Supports Enterprise Integration Patterns

- Supports all the integration patters from the different pattern categories such as Messaging Systems, Messaging Channels, Message Constructions, Message Routing, Message transformation, Messaging endpoints, System management

Deliver all required Enterprise Service Bus features/Core functionalities

- Delivers all the core functionalities of ESB such as Web services, Message transformation, protocol mediation, Message Routing, Location Transparency, Transport Protocol Conversion, Message enhancement, Security, Monitoring and management

Offers a complete and cohesive SOA Platform

- Offers complete SOA platforms that includes ESB, Message Broker, Governance Registry, Business Process Server, Data Services Server, and Application Server

Core performance and quality of service capacity

- Have the quality of service capability with reliability, availability, scalability

Graphical Management and Monitoring Capabilities

- Offers a good graphical management and monitoring feature

Freedom and Flexibility of using the Source Code

- Provides the freedom and flexibility to use the full source code freely with no restriction

Offer a Graphical ESB Development Workbench

- Provides easy to use graphical ESB development environment

Open Business Model

- Offers free product, open source, open content, open tools for development which is based on open standards

Security and Identity Management

- Offers an identity and access management plat form with a security features that provides authorization, authentication and encryption functionality

Ease of Installation

- Easy to install with simple procedure in any platform

Freely Available support

- Availability of free Code Base Examples, Active community, extensive documentation, online documents, videos, 24X7 support

Ease of Development

- Availability graphical environment for development that rapidly create and orchestrate services, complex integration patterns can be implemented easily, requires only the skill of an analyst, not a programmer. Availability of development, testing and deployment of the middleware application.

No Commercial Version (Enterprise Version) Available

- Is there a commercial version or enterprise version available for the open source product? If there is a commercial product available there is a big chance for the open source components missing that could lead to purchase the commercial version

Less Cost

- No or less cost for Licensing, Operation management and development expense

Build skilled local manpower

- Ethiopia is one of the developing countries and it is necessary to be able to build skilled professionals as part of the organizations. So an ESB product that is easy to learn with full of freely available support system is necessary. It is also mandatory that all the development tools and environments for the ESB product to be easily available and understandable. All the features that the ESB product provide and future to come must be guaranteed to be available free of cost so that additional features can be easily customized by the local professionals.

Matches with current organization plan

- The organizations that participate in the integration are service providing organizations and there is a need to facilitate these services. Currently ERCA is planning to integrate these services by using a single window. Therefore the ESB should comply with the current plan of ERCA by supporting the work plan with lesser cost and long lasting solution.

The comparison matrix result is completed based on the research papers reviewed and the current situation and protocols of our country Ethiopia and the organizations involved on the study.

Table 2.2 illustrates the comparison matrix and their evaluation result.

3 – Fully satisfy the criteria

2 – Satisfy the criteria at a moderate level

1 – Satisfy the criteria at a limited level

0 – Does not full fill the criteria

Table 2.2: Open Source ESBs Comparison Matrix

No.	Criteria	WSO2 ESB	Mule ESB Community	JBoss ESB	Apache ServiceMix
1	Supports Enterprise Integration Patterns	3	3	3	3
2	Deliver all required Enterprise Service Bus features/Core functionalities	3	3	3	3
3	Offers a complete and cohesive SOA Platform	3	0	3	0
4	Core performance and quality of service capacity	3	2	2	3
5	Graphical Management and Monitoring Capabilities	3	0	3	0
6	Freedom and Flexibility of using the Source Code	3	2	2	2
7	Offer a Graphical ESB Development Workbench	3	3	3	1
8	Open Business Model	3	3	0	3
9	Security and Identity Management	3	1	1	3
10	Ease of Installation	3	3	3	0
11	Freely Available Support	3	2	2	2
12	Ease of Development	3	2	3	0
13	No Commercial Version Available	3	0	0	0

14	Less Cost	3	1	1	1
15	Build skilled local manpower	3	1	2	2
16	Matches with current organization plan	3	2	2	2

2.2.2.5 Open Source ESBs Comparison Analysis Result

After summarizing the result from the above comparison matrix table, it is reached to the conclusion that for this work the suitable open source ESB product to implement the integration solution for the problem under study to be WSO2 ESB.

In general WSO2 ESB has good documentation, lots of samples, tutorial (video, text), articles and case studies available. It also has a very active community support and all development tools are available freely and are easy to use. Therefore we have selected the WSO2 ESB for developing the demonstration for this work.

2.3 Difference between EAI and ESB

A lot of the early ESB products had a history in the enterprise application integration market. It was sometimes hard to tell the difference between some ESB products and their EAI predecessors.

As explained in [12] there are two main differences between EAI and ESB product. The first difference between the two is the change from the hub-and-spoke model in EAI products to a bus-based model in ESB products. The hub-and-spoke model is a centralized architecture, where all data exchange is processed by a hub, or broker. The hub-and-spoke model can be seen as the successor of the point-to-point model. The bus model, on the other hand, uses a distributed architecture, in which the ESB functionality can be implemented by several physically separated functions.

The second main difference between EAI and ESB products is the use of open standards. EAI products like WebSphere Message Broker, TIBCO BusinessWorks, and Sonic XQ were mainly based on proprietary technology to implement messaging functionality and transformation logic. ESB products are based on open standards, such as Java Message Service, XML, J2EE Connector Architecture (JCA), and web services standards [12].

Chapter 3

Related Works

There are variety of research papers and works done on system integration using Enterprise service bus. In this chapter description of works related to this project will be presented.

Research by Utomo and Wellem [18] successfully integrated graduation business process that involves some units including Faculty, Library, Dormitory, Administration and Academic Bureau, Student Affair bureau, Finance Department, and Laboratory. This integration of graduation business process uses ESB as integration middleware.

The heterogeneity of business and information system with respect to hardware and software platform is common in almost all company or organization. The effort to deal with the problem is usually unsuccessful because of the complexity of technology information architecture from a heterogenic application built from different architecture, programming language, and platforms and the existing applications have to keep running when it is improved. To integrate business and information system, an integration that also serves as mediation between the two layers is needed. The authors in their research used OpenESB tool as middleware ESB infrastructure.

The three roles of ESB in terms of routing, protocol transformation and message/data transformation has been performed successfully in the research. JBI module produced by the research has proven the use of ESB to route service users to local and external service provider. Protocol transformation has also been performed successfully through the JBI module. JBI module reveals HTTP inter-protocol communication using JDBC, between HTTP and SMTP, or between HTTP and FILE. Message transformation can be done through XML data format.

The Graduation business process involving the variety components successfully integrated using OpenESB integration middleware. The use of ESB in this integration process solves the complexity of information technology architecture from heterogenic applications built from different programming architecture and language as well as different platform.

Another related work reviewed is the one presented on the paper [16] titled, *the research and implementation of power application system integration based on enterprise service*

bus. In the research paper, the working principle of enterprise service bus and integration architecture was discussed and the system integration was implemented using the enterprise service bus based on analyzing the typical business scenario of electric power.

According to electricity enterprise business applications and the construction of Enterprise Resource Planning software, the pattern of integration platform is mainly used to achieve common integrated platform-based application. The authors of this article used the Oracle Service Bus for the integration.

The business scenario in this research is the enterprise resource management system (Enterprise Resource Planning (ERP)) and production management (PMS) system information synchronization between the equipment accounting businesses. PMS system in the equipment is the basis of network production operation management. Equipment management is to achieve kind of basic information on equipment, technical parameters, operation and maintenance of information and equipment change of management; ERP system master data in the device is to realize main equipment and assets' correspondence, which is the bridge of the PMS system equipment and the ERP system master data. ERP system master data uses the equipment accounting interface integration, realizing assets' whole life cycle tracking of the physical changes and equipment maintenance Cost Absorption and assets of the account, card, and material consistency.

Through the interface, the equipment accounting in PMS system is shared by the data center area and ERP data, to prevent users maintaining data duplicatedly between PMS system and the ERP system and maintain data consistency.

The realization of PMS and ERP device management device in the process of master data integration, the ERP related Web Services are provided and registered to the enterprise service bus. Production management sends messages to the JMS server and achieves the relevant service calls by the ESB.

Systems integration based on Enterprise Service Bus achieve the transporting of information and instructions between different applications safely and efficiently, which provide comprehensive information from cross-sectoral, cross-system, cross-application to enable data between applications flow smoothly and reduce the business

processing time, also, increase efficiency greatly. The result of the research showed that the data was received and applied smoothly among systems, the processing time was reduced, and the efficiency and accuracy of the business operation were improved greatly.

Another related work on applying Enterprise Service Bus for integration is a research on remote sensing data service system based on Mule ESB [19]. The paper proposes the model of remote sensing data service system based on Enterprise Service Bus to realize the sharing of data and service. The authors planned to build a sharing and public platform to realize the interoperation of remote sensing data among different systems. By applying the Service Oriented Architecture to reconstruct the system, the approach abstracted the function models from the system as services which are connected by well-defined interface or contract. The interfaces do not depend on programming language, operating system or platform. The interoperable platform for the integration of services by the heterogeneous applications within service oriented architecture is provided by an Enterprise Service Bus. The research displays the model of open Remote Sensing Data Service System and uses Mule ESB open source product to complete the design of the model.

The overall purpose of the system is to manage the distribution of data in remote sensing. The data in remote sensing is complicated with abundant information. The special information systems are based on specific EsriArcGis, MapInfo platform leading to the tight-coupling phenomenon. In order to realize the loose coupling of services and platform, they try to adopt the new architecture which is SOA.

Mule ESB community release is used for this research and to integrate the external applications, web services are used. By combing Mule ESB and Web Service as a core idea to design the spatial information system, the authors encapsulate the GIS applications of the systems as Web Service and integrate these services via Mule ESB.

The overview of the process flow of the model is, a customer signs in the portal to send request to Mule ESB which can get through proxy to look for the services which satisfies the customer’s need. After discovering the GIS service, Mule will invoke the corresponding services with inputting parameters. Then the GIS application system accesses to the spatial database or ordinary to get available data. At last, customer receives the retuned data.

During the system runs, Mule ESB will authenticate the customer's identification, discover the service meeting customer's need, routing, transforming and dealing with the messages.

The research provides a model and puts the model in to practice, it is expected that a system to get agile. The purpose of this work in general is making full use of remote sensing resources and by through this model the customer can get access to more data information easily.

By taking the characteristics of ESB and the primal principle of ESB in to account, a novel idea of this technology is presented, and applied to a School Common Data Platform in the research work [20]. The research is done on Shanghai Second Polytechnic University school common data platform for specific applications. The university has nine campuses and it needs information management mechanism [20]. In order to address the application of existing subsystems integration and communication problems and to enable the whole system of information application to become an organic whole, the system uses ESB technology to manage multiple services.

The web based ESB integration in the school common data platform is the way of using web based ESB integration within the organization not only between the legacy systems, legacy systems with the new deployment of interoperable systems, as well as integrated application systems organization the application and the organization of interoperability between the systems. The use of ESB and SOA helps in reducing the coupling between the systems and improves the system and the system robustness.

The security considerations of the web based ESB has a prominent place. The security measures that are taken for the system are: since the system is in the internal campus network, the network has its own security mechanism to protect it from illegal users of external attackers. Another measure is used is use of management interface for ESB Service center that has the capacity to authenticate, authorize and audit. This helps to avoid illegal bypass ESB Customer Service Center. The last security measure taken is use of Secure Socket Layer (SSL) to achieve end-to-end integrity and confidentiality of information between service requesters and web Servers communication. Finally the authors experiment the feasibility and the result shows the system performance is at good efficiency and it met the actual needs.

An open source implementation of Enterprise service bus project for eBay is another related work on use of ESB for integration [17]. The integration is developed by WSO2. eBay is the world's largest online marketplace. Today, more than 94 million active users around the globe flock to eBay to find the best deals in cyberspace. In 2010 alone, the total value of goods sold on eBay was a staggering $62 billion or $2,000 every second.

Along with eBay's success comes a huge demand to ensure reliable, 24x7 availability of the services that enable these transactions. There's no room for error, especially during the peak online holiday shopping season. For eBay, that has meant using the WSO2 Enterprise Service Bus (WSO2 ESB) to carry more than 1 billion transactions per day during peak shopping times in 2010.

As described on case study by WSO2 on eBay's use of their open source product in Enterprise service Bus [17], eBay used WSO2 ESB for handling the middleware requirements of its online marketplace. The WSO2 ESB outperformed all other software options in both speed and reliability. The WSO2 ESB demonstrated the flexibility to grow and adapt to eBay's evolving requirements for handling transformations, orchestrations, and complex message flows.

Each pool of high-end WSO2 ESB servers is provisioning one of three use cases that support eBay's various business functions: routing, orchestration, and service chaining. These deployments currently include dozens WSO2 ESB instances, which altogether accommodate external and internal traffic loads from different functional areas such as shopping, trading, checkout, and mobile, to name a few.

The resource utilization of the WSO2 ESB is very efficient, allowing massive deployments to run on a minimum amount of servers, saving time and money for the customers. Additionally, the memory usage of the WSO2 ESB instances remains stable irrespective of the traffic load fluctuations at eBay to ensure high availability.

Because the WSO2 ESB instances easily inter-operate with eBay's in-house and third-party monitoring systems, they have helped to improve the overall monitoring capabilities of the system.

With the use of WSO2 ESB deployments, eBay successfully provided a reliable and an efficient shopping experience to its vast customer base worldwide, much to the delight of the eBay's management team and the customers.

Between December 2010 and January 2011, Forrester conducted briefings and demonstration reviews with five commercial and four open source ESB product vendors: FuseSource, IBM, MuleSoft, Oracle, Progress Software, red Hat, Software AG, Tibco Software, and WSO2. Interviews with more than twenty customers who are currently using the evaluated products were conducted and hundreds of customers were spoken related to their ESB implementation activities.

To assess the state of the enterprise service bus market and see how the vendors stack up against each other, Forester [24] evaluated the strengths and weaknesses of top ESB vendors. Forrester used a combination of three data sources to assess the strengths and weaknesses of each solution: Vendor Survey, Product Demonstration and Customer Reference Calls. Forester developed a comprehensive set of evaluation criteria and evaluated vendors against 109 criteria, which is grouped into three high-level buckets: Current Offering, Strategy and Market presentation.

In current offerings category the breadth of each vendor's ESB offering across 79 criteria, including the major categories of architecture, orchestration, mediation, connection, and change and control is examined. In strategy, the strength of each vendor's strategy across 15 criteria, including product strategy, solution cost, strategic alliances, and customer reference checks were evaluated. And in the final category, market presentation, each vendor's penetration in the ESB market is evaluated using 15 criteria, including installed base, new customers, annual ESB revenue, and delivery footprint.

The ESB Forester Wave evaluation spreadsheet explores all the functional areas of ESB in depth as well as scores evaluated vendors' capabilities within each. And after a deep evaluation and analysis of the products it reaches to a result of conclusion that shows WSO2 is a leading open source ESB product. FuseSource and WSO2 scored highly in most of the evaluated areas; each of these vendors' products represents a solid ESB solution that would be a good choice for meeting many enterprise integration and service-oriented architecture requirements.

Yelno on its whitepaper titled "*ESB comparison*" [25] describes what is needed to know to evaluate and select the right ESB for business. It explains the differences between the ESBs on a feature level.

After an investigation, endurance test and real-life practice the whitepaper resulted with the comparison matrix that compares the five ESB solutions: WSO2, Mule, FuseSource 7.0.0 JbossTibco AM based on basic and important ESB features: Enterprise integration patterns, delivers all required ESB features, complete SOA platform, SOA Governance, graphical ESB development, architecture, cloud integration platform, cloud connectors, legacy adapters, performance, security and identity management and open business model.

As shown on the feature comparison result table on the paper, WSO2 offers a complete and flexible Enterprise Service Bus solution. The paper also indicates currently there are many companies that chose the WSO2 ESB for their integration solution as their technology of choice to solve their business challenge namely eBay, Volvo, Boeing, SUVA and so on[7].

Yenlo highly recommended choosing the WSO2 Enterprise Service Bus for the reasons that WSO2 qualifies for what is required of from an Enterprise Service Bus that is:

- Is base on Java open source;
- Is based on open standards;
- Is extremely reliable;
- Is extremely fast;
- Is a full enterprise solution with all the components that can be wish for;
- Is fully supported including 24/7 managed services;
- And does not cost a lot of money to implement.

The article [26] presents the latest performance study conducted to compare the performance of the latest release of WSO2 ESB i.e. WSO2 ESB 4.8.1 with a number of leading open source ESBs. The open source ESBs considered in this study are: WSO2 ESB 4.8.1, AdroitLogicUltra ESB v2.0.0 – Enhanced, Mule ESB Community Edition v3.4.0 and Talend ESB SE v5.3.1.

[7] http://wso2.com/about/customers/

The performance test carried out in this analysis report is on machines running in Amazon EC2 (Amazon Elastic Compute Cloud). Amazon EC2 is a web service that provides resizable compute capacity in the cloud.

The result of the performance test conducted on the four open source ESBs for the different proxy services: *DirectProxy*, *CBRProxy*, *CBRSOAPHeaderProxy*, *CBRTransportHeaderProxy*, *XSLTProxy* and *SecureProxy* reported in table and graph on the article [26]. The result [26], indicate thatWSO2 ESB 4.8.1 outperforms all other compared open source ESBs.

A research work on ESB product evaluation by Sanjay and Amit on [30] evaluated three open source ESBs and compared them both qualitatively and quantitatively. The empirical results were statistically tested to determine the statistical significance of the result. The open source ESBs selected for study and evaluation were Mule, WSO2 ESB and ServiceMix. The intention of the study was to evaluate the core features of the ESB according to a metrics.

The core functionalities considered are Virtualization, Content Based Routing and Mediation. For the study, metrics were captured to evaluate the ESBs on each of these three core features. To test load-handling and scalability for each scenario, two sets of tests are conducted: Multiple clients sending a fixed payload of 100 KB with the number of clients tested 1, 20, 40, 80, and 160 and a single client sending an increasing payload with a payloads 1 KB, 10 KB, 50 KB, 100 KB, 500 KB, and 1 M.

The metrics that are considered in the study are: Mean Response Time which is the amount of time elapsed from the moment the request was sent to the time a reply was received and Throughput i.e. number of transactions per second.

In order to obtain the best results, each test was run a minimum of 10 times each. Testing was done using Grinder, an open source stress testing tool.

The results of all the three scenarios tested with the two test sets to analyze the metrics were reported in a graph on the study, which gives a clear picture of the ESBs performance for the given metrics. A statistical analysis were also conducted on the data collected using Student's T-Test on each metric.

The result of the analysis shows that ServiceMix and WSO2 handled scalability better than Mule in Direct proxy scenario.WSO2 performed better than Mule in loading handling and scalability for the content-based routing scenario. ServiceMix handled scalability better than WSO2 on direct proxy scenario and in the transformation proxy scenario.

A set of criteria for subjective assessment of the ESBs were also set in the study. The subjective assessment includes: installation, Code base/ Examples, ease of development, features, API/ Documentation, online help and community/ forums. The ESBs were assigned a score for each criterion and the results were recorded. The result illustrated that WSO2 performs better than the other two open source ESBs in the subjective assessment followed by Mule.

The thesis work titled, *Requirements for an Enterprise Application Integration tool* [27], presents four selected integration tools: IBM WebSphere Message Broker, Mule enterprise and community, WSO2 and Apache ServiceMix. Each tool was installed for researching and testing. The summary of the tools under study in the work are summarized in the following paragraphs.

The Mule ESB has two versions: Mule enterprise and Mule community. As described on [27] Mule enterprise version is a commercial product with more features that provides better reliability, security, performance, control and message transformation. It includes a graphical web based management console. Mule ESB Community Edition is a stripped down version of the commercial version missing some important features. It lacks the graphical management console but is completely free to use. The community version also lacks some special proprietary connectors like SAP and IBM WebSphere connectors. As the community version does not have the Mule Enterprise Security package which is included in the commercial version it is not as secure. Performance and reliability of the community version has also been greatly reduced by omitting the high availability and caching features. ActiveMQ based on JMS protocol is most widely popular message bus. Nevertheless, Mule ESB community release does not support ActiveMQ [19]. There is also no function available to support dynamic discovering services from registry in Mule ESB community version [19]. There is less options for performance management in community version than what is included in the commercial version. Also many analysis tools have been omitted

that would help problem resolution and the support is only limited to the community forums. Lack of so many features basically makes the community mule somewhat a crippled one [27]. Unfortunately, as listed above, all the most important features are inside the commercial version and it seems that MuleSoft is pushing the commercial version to customers quite heavily so the open source version is left missing on some important features. Deciding to use this tool would require commitment on taking the commercial route [27].

WSO2 ESB is 100% open source and there is no commercial version [27]. WSO2 has a good looking and extensive web based management console which is not so common on open source tools. Inside the management console is basically included everything from message routing and transformation to server management and configuration [27]. The community around WSO2 ESB seems quite active and there is extensive documentation on how to get started [27].

Apache ServiceMix is a free open source implementation of an Enterprise Service Bus licensed under Apache license v2. There is a commercial Enterprise Service Bus implementation called Fuse ESB that is heavily based on ServiceMix and can be thought of as the commercial version of ServiceMix [27].

The related research works reviewed in this chapter helps in grasping knowledge in the contribution of ESB based integration approach for different environments. It also becomes the key input for constructing the open source Enterprise Service Bus products evaluation criteria and evaluation matrix. The result for the criteria on the evaluation matrix on Chapter four is also gathered based on the result of the research documents by different professionals and organizations reviewed here in this chapter.

Chapter 4

Existing System Analysis for Declaration of Imported Vehicles

This chapter briefly describes the existing system for vehicle import procedure. The requirement is collected by conducting document analysis and through interview. Documents that are relevant for this study are revised and to fill the gap an interview is conducted. The Software program coordinator from ERCA, an individual from Transport authority, an individual from NBE and a vehicle importer, whom their names not mentioned for privacy purpose, were communicated to gather additional information. Findings on the existing system and business processes on import procedure for vehicles are described in detail in the following subsections.

Import procedure refers to the processes involved and undertaken by an importer and/or agent during importation of goods into Ethiopia. Depending on the type of commodity, different requirements and procedures are required to be completed by the importer.

For importing a vehicle, the importer must have valid business license from Ministry of Trade, a bank import permit from National Bank of Ethiopia and vehicle Import Permit from Ministry of Transport.

4.1 Vehicle Import Permit

Importing of vehicles requires a registration and authorization from the Transport Authority. The vehicle import permit process is applied for each import and covers all types of motor vehicles; new, used, vehicles imported by investors and embassies etc.

The importer applies for permit to the Transport Authority for importing vehicles. To proceed with the permit processing, business license from MoT, Performa invoice and packing list are required. Detail information about the imported vehicle should be provided in addition to the above documents. Such as:

- Imported vehicle standard inspection form;
- Forms for technical specifications (two forms in Amharic to collect technical information about the vehicles);
- Performa invoice;

- If an Ethiopian national returning to Ethiopia: a letter from the Ethiopian Embassy abroad attesting the stay outside and providing detailed information about the car;
- If investor, an investment permit issued by EIA;
- If Embassy: letter from the Embassy with detailed information about the car and a letter from the Ministry of Foreign Affairs;
- If NGO's: letter of support from the organization with detailed information about the car;
- If religious organization: letter from the religious organization with detailed information about the car ;
- If companies importing a car or cars temporarily for specific purposes: agreement or contract with the Ethiopian company with detailed information about the car(s);
- If importers of new or old cars: application form, business license (MOT) and a vehicle competency certificate;
- If individuals: application form and letter confirming for example that the vehicle was given following some sport event (as award), detailed information about the car and a bill of loading.

The documents verified for its validity and Transport Authority approves, rejects or requests additional information. If approved Transport Authority provides Import permit for importing vehicles to the importer. This process is paper based and follows manual procedures.

4.2 Bank Import Permit

All international trade operations must be conducted through authorized banks and a foreign exchange permit must be obtained. All commercial banks processes linked to foreign exchange transactions are still mainly manual. 80% of bank payments are made using an L/C in Ethiopia. L/Cs contains information as requested by the issuing commercial bank. In the case of L/Cs issued in Ethiopia, a confirmation process is generally required by the foreign correspondent bank.

Before the import process, the importer has to apply for a bank import permit for foreign exchange. To proceed with the bank import permit process, the importer is required to have necessary documents i.e. valid business license, Performa invoice and import permit for the

vehicle from Ministry of Transport. Performa Invoice or contract from the suppliers includes information about the type and quantity of the commodity, price per unit, FOB amount, fright if any and other charges. Insurance arrangement is made locally.

Payments for imports can be made by letter of credit (L/C), cash against documents (CAD) or advance payment. Approval of an application for foreign exchange is subjected to the following conditions [43]:

- The product to be imported must be free from any import prohibition
- the application for foreign exchange shall be:-
    - Properly completed, signed and sealed with the office stamp of the importer in sufficient copies,
    - Supported by manufacturers' or suppliers' invoices in sufficient copies showing clearly full description of the goods including quantity, grade, quality, volume, measurement, weight, and unit and total price of the goods at a named place of delivery;
- Terms of payment must be clearly shown in the invoice;
- Evidence must be produced that adequate insurance cover has been arranged with Ethiopian Insurance Corporation particularly for goods imported under letters of credit;

Modes of payment allowed for import are Letter of credit, Cash against document and advance payment. Letter of credit is a written commitment to pay, by a buyer's or importer's bank to the seller's or exporter's bank. It guarantees payment has been made to the suplier. A supplier or exporter that has sold to an Ethiopian Importer on L/C basis should present to the bank these four sets of documents: Bill of Lading, Shipping Document, Commercial Invoice, Packing List and Certificate of Origin.

4.3 Customs Import Declaration

The Importer or the authorized Clearing Agent applies to ERCA for import declaration for imported vehicles clearance. The Importer or the authorized Clearing Agent is requested for important documents, which are, business license, Bank Permit and Vehicle Import Permit. ERCA validates the application information against the documents provided. If approved, the import declaration registered and the amount of duties and taxes will be calculated.

4.4 Imported Vehicle Declaration Scenario

4.4.1 Scenario Description

The scenario to be demonstrated in this work is the business process that is important for an imported vehicle declaration process. To declare an imported vehicle at ERCA, import permit from MoTr, Bank import permit and a valid import license are required. After checking the validity of the above documents from the different organizations, the HS_Code[8] for the imported vehicle will be retrieved from the database by taking the description of the vehicle. After retrieving the HS_Code, the corresponding tax rates for it will be retrieved by using the HS_Code. Finally the payable tax amount for the imported vehicle will be calculated by taking the tax rates, the cost for the vehicle, the fright amount, insurance amount and additional cost values, if any, in to consideration. This scenario will be demonstrated by integrating the various systems from different organizations by using an ESB.

4.5 Imported Vehicles Declaration Use Case Diagram

The following use case diagram, *Figure 4.1*, demonstrates the scenario for declaring an imported vehicle process.

[8] Harmonized System Codes (HS_Code) is a standard issued by the World Customs Organization (WCO) to unify the classification of the goods. These are six digit codes for identifying different products across the world. The country using these HS_Codes can suffix additional digits to the existing six digits according to their needs.

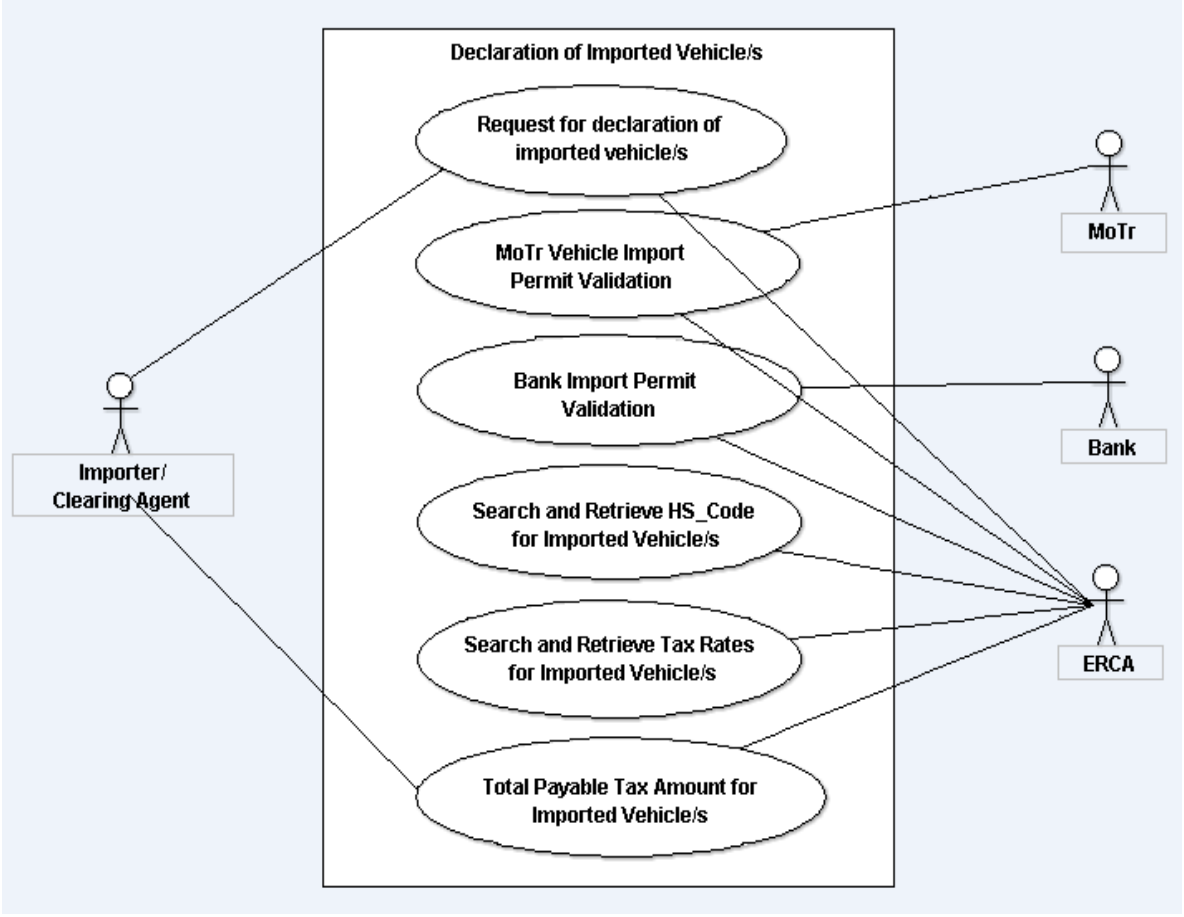

Figure 4.1: Use Case Diagram for Imported vehicle declaration

Main Flow of the Declaration of Imported Vehicles Process:

1. Request for declaring Imported vehicle/s is sent to the system by an importer or an agent with all the necessary information about the vehicle/s imported which are importer information, description of the vehicle, cost of the vehicle, freight cost, insurance cost, additional cost and number of vehicles imported
2. The system checks whether the vehicle imported has a permit from the Ministry of Transport
3. If the imported vehicle/s has an import permit from Ministry of Transport, the system checks whether it has a bank permit or not
4. If the imported vehicle/s has both permit from Ministry of Transport and Bank, the system will take the details of the vehicle and searches and retrieves the HS_Code

5. The system searches and retrieves the corresponding five tax rates i.e. Duty rate, Excise rate, VAT, Sur rate and Withholding rate for the vehicle/s with HS_Code retrieved
6. The system will calculate the total payable tax amount for the vehicle/s by taking the detail information about the imported vehicle/s and the corresponding tax rates

4.5.1 Use Cases Description

In this subsection the description of the use cases in the use case diagram above is presented. The use cases are:

- Use Case 1 – Request for declaration of imported vehicle/s
- Use Case 2 – MoTr Vehicle Import Permit Validation
- Use Case 3 – Bank Import Permit Validation
- Use Case 4 – Search and Retrieve HS_Code for Imported Vehicle/s
- Use Case 5 – Search and Retrieve Tax Rates for Imported Vehicle/s
- Use Case 6 – Total Payable Tax Amount for Imported Vehicle/s

Use Case Name: Request for declaration of imported vehicle/s

Use Case Description: This use case describes the action for the system (ESB) to accept request for an imported vehicle/s declaration from an importer or clearing agent

Actors:

- Importer or Clearing Agent
- ERCA

Pre-conditions:

- The ESB is up and running

Normal Flow:

1. Importer/Agent send request for declaration of imported vehicle to the ESB
2. The ESB checks all the necessary data is provided
3. The ESB encode the input data
4. The ESB split the data and forward the data to the appropriate endpoints of the service providers at ERCA

Post-conditions:

- The request message is delivered to the proxy service of the ESB

Use Case Name: MoTr Vehicle Import Permit Validation

Use Case Description: This use case describes the action that a formatted message with encoded data correctly routed to the service provider responsible for this request data by the routing component of the ESB and the corresponding Vehicle Import Permit status forwarded back to the ESB

Actors:

- MoTr
- ERCA

Pre-conditions:

- Successful completion of Use Case 1 as described above

Normal Flow:

1. The ESB forwards service request about the MoTr import permit to the correct endpoint
2. MoTr checks the status of the import permit for the requested data and respond the status
3. The endpoint accepts the result and forward it to the ESB's Proxy server

Post-conditions:

- The ESB receives the status of the Ministry of transport permit from the system at MoTr

Use Case Name: Bank Import Permit Validation

Use Case Description: This use case describes the process of a formatted message with encoded data correctly routed to the service provider responsible for this request data by the routing component of the ESB and the corresponding Bank Import Permit status result is forwarded to the ESB

Actors:

- Bank
- ERCA

Pre-conditions:

- Successful completion of Use Case 1 and Use Case 2

Normal Flow:

1. The ESB forwards service request about the Bank import permit to the correct endpoint
2. Bank checks the status of the import permit for the requested data and respond the status
3. The endpoint accepts the result and forward it to the ESB's Proxy server

Post-conditions:

- The ESB receives the status of the Bank Import Permit from the system at the Bank

Use Case Name: Search and Retrieve HS_Code for Imported Vehicle/s

Use Case Description: This use case describes the process of a formatted message with encoded data correctly routed to the service provider responsible for HS_Code retrieval by the routing component of the ESB and the corresponding response which consists the HS_Code value is forwarded to the ESB

Actors:

- ERCA

Pre-conditions:

- Successful completion of Use Case 1, Use Case 2 and Use Case 3

Normal Flow:

1. The ESB filters the data from the request important for HS_Code retrieval and transform to the appropriate format
2. The ESB forwards formatted service request for HS_Code of the imported Vehicle to the corresponding endpoint
3. ERCA Searches for the HS_Code for the requested data

4. ERCA returns the HS_Code for the requested data
5. The endpoint accepts the result and forward it to the ESB's Proxy server

Post-conditions:

- The ESB receives the HS_Code for the imported vehicle/s from the responsible system at ERCA

Exceptions

- The system returns the HS_Code not found message for requested data

Use Case Name: Search and Retrieve Tax Rates for Imported Vehicle/s

Use Case Description: This use case describes the process a formatted message with encoded data correctly routed to the service provider responsible for Tax Rate retrieval by the routing component of the ESB and the corresponding tax rates forwarded back to the ESB

Actors:

- ERCA

Pre-conditions:

- Successful completion of Use Case 4

Normal Flow:

1. The ESB filters the data from the request important for Tax Rate retrieval and transform to the appropriate format
2. The ESB forwards formatted service request for Tax Rates of the imported Vehicle to the corresponding endpoint
3. ERCA Searches for the Tax Rates for the requested data
4. ERCA returns the Tax Rates for the requested data
5. The endpoint accepts the result and forward it to the ESB's Proxy server

Post-conditions:

- The ESB receives the Tax Rates for the imported vehicle/s from the responsible system at ERCA

Use Case Name: Total Payable Tax Amount for Imported Vehicle/s

Use Case Description: This use case describes the process a formatted message correctly routed to the service provider responsible for Total Payable Tax Amount Calculator by the routing component of the ESB and the total payable tax amount is forwarded back to the ESB

Actors:

- ERCA
- Importer/ Clearing agent

Pre-conditions:

- Successful completion of Use Case 1 and Use Case 5

Normal Flow:

1. The ESB filters the data from the request important for Total Tax amount calculation and transform it to the appropriate format
2. The ESB forwards formatted service request for total Tax amount of the imported Vehicle/s to the corresponding endpoint
3. ERCA Process the request and return the result to the ESB
4. The endpoint accepts the result and forward it to the ESB's Proxy server
5. The result is forwarded to the client that originate the request
6. The importer or the clearing agent is notified the total tax amount for the imported vehicle/s

Post-conditions:

- The ESB receives the Total Payable Tax amount for the imported vehicle/s from the responsible system at ERCA and notify the importer or clearing agent

Chapter 5

ESB Design and Implementation

This chapter refines the requirements described in Chapter Five in to a design that provides the basis for the implementation and the implementation of the integration. It will start with the overall architecture for the scenario described on the previous chapter and follow by a detail design and implementation of the WSO2 ESB.

5.1 ESB Design

The ESB of this project is designed for integration of five systems at three different organizations. The systems to be integrated are:

- System that provides the status of Transport Authority vehicle import permit,
- System that provides the status of the Bank import permit for the vehicle imported,
- System that provides the HS_Code for the vehicle imported,
- System that provides the different tax rates for the vehicle imported,
- System that calculates the total payable tax for the vehicle imported

For the scenario, it is taken the assumption that the vehicle import permission process is taken place and the status of the vehicle import permission is available at the Transport authority. And the same is true for the bank import permit. The scope of the scenario is limited to checking the validity of the import permits from the two organizations only; the detail business process that are undertaken by the organizations that leads to the decisions to grant or deny the permit are not included. The scope of the scenario also limited to the declaration process of the imported vehicle from ERCA. It ranges from validating the documents and the preconditions fulfillment up to the calculation of the total payable tax.

The following diagram, *Figure 5.1*, illustrates the different systems that will be involved in providing integrated vehicle declaration system and the WSO2 ESB as central point in the integration.

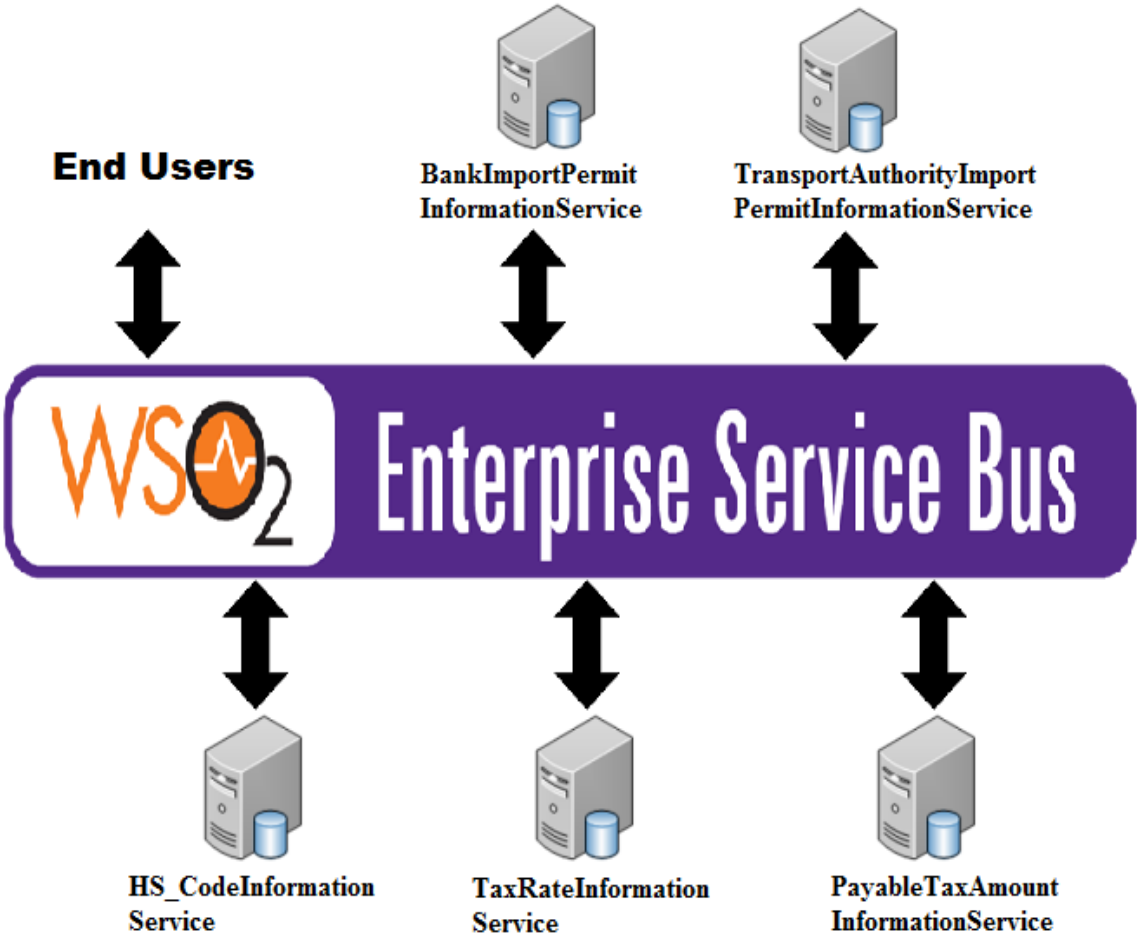

Figure 5.1: Imported Vehicle Declaration Process Service Integration with WSO2 ESB

The entire process flow of the imported vehicle declaration is as shown in the business process diagram *Figure 5.2*.

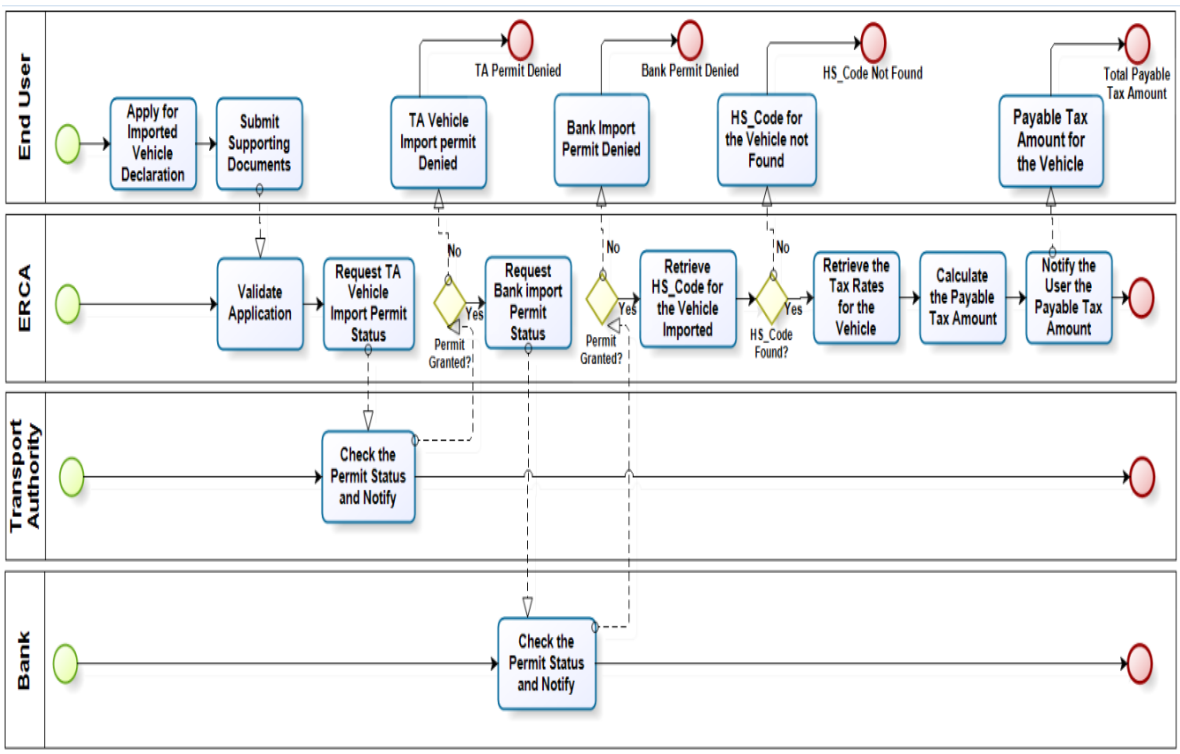

Figure 5.2: Business process diagram for imported vehicle declaration process

As illustrated in *Figure 5.2*, the business process of declaring imported vehicles starts when an Importer/Clearing Agent apply for declaration. To proceed with the processing, the Importer/Clearing Agent requested to submit the supporting documents. ERCA then validate the application and request for vehicle import permit status from the Transport Authority. If the permit is granted, it will request for the Bank import permit status. If the status is granted, it will retrieve the corresponding HS_Code value for the vehicle. Finally the tax rates for the vehicle will be retrieved and the tax amount to be paid will be calculated and the Importer will be notified. If the result of the request for the vehicle import permit status and the bank import permit status is denied, the user will be notified to process the two permits prior to the declaration process.

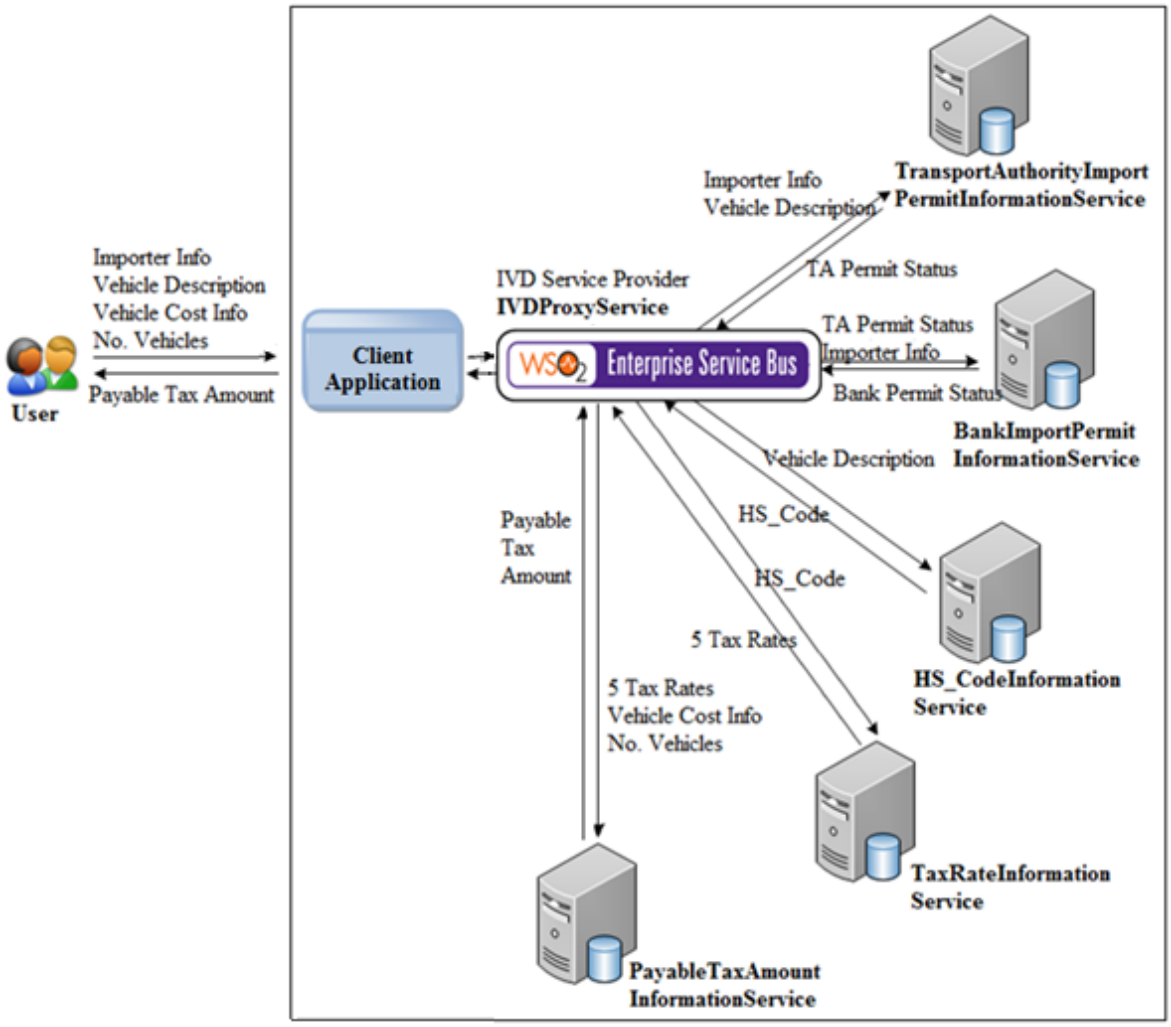

Figure 5.3: Import Vehicle Declaration Scenario – Summary

As shown in the *Figure 5.3*, in the import vehicle declaration scenario there are five service calls involved. The user sends the request to a proxy service that will perform service orchestration. The proxy will first invoke *TransportAuthorityImportPermitInformationService* with importers information and vehicle description and obtain the Transport Authority import permit status. The vehicle description should include engine capacity, status (used/new) and purpose of the import (public transportation, NGO, investment, Embassy, religious organization, importer, etc.). It will then invoke *BankImportPermitInformationService* with the Transport Authority import permit status, importers information and vehicle description and obtain Bank permit status. Then after it will invoke the *HS_CodeInformationService* with the vehicle description and

obtain the HS_Code value. The proxy service then will obtain the five tax rates (Duty rate, Excise rate, VAT, Sur rate and Withholding rate) value for the vehicle by invoking the *TaxRateInformationService* with the HS_Code. Finally, it will invoke *PayableTaxAmountInformationService* with the five tax rates, vehicle cost information (cost, insurance and freight) and the number of vehicles imported and will return the payable tax amount; this payable tax amount will be sent to the user.

The vehicle declaration process that integrates the five systems described above implemented using WSO2 ESB as shown in *Figure 5.4*.

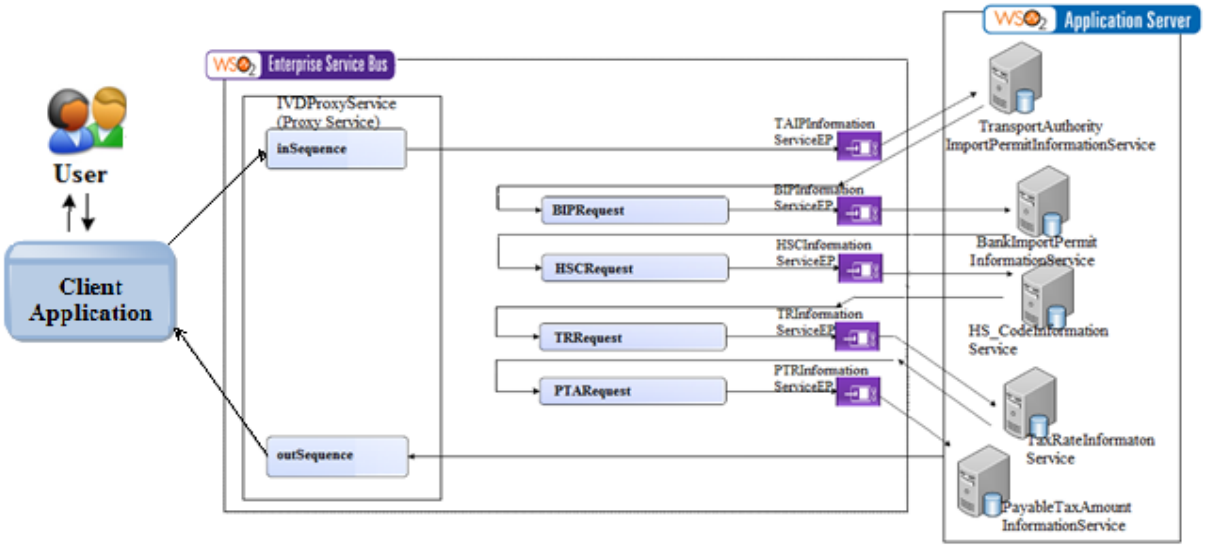

Figure 5.4: Vehicle declaration business process implemented using WSO2 ESB

In the *Figure 5.4* the following takes place in the given order:

1. The request is first received by the inSequence of the *IVDProxyService* Proxy service and mediation occurs
2. The message is forwarded to the *TransportAuthorityImportPermitInformationService* service via the *TAIPInformationServiceEP* end point
3. The response (Transport Authority import permit status) from *TransportAuthorityImportPermitInformationService* is forwarded to *BIPRequest* sequence for mediation
4. *BIPRequest* forwards the message to the *BankImportPermitInformationService* service via *BIPInformationServiceEP* endpoint

5. The response (Bank permit status) from *BankImportPermitInformationService* is forwarded to *HSCRequest* sequence for mediation
6. *HSCRequest* forwards the message to *HS_CodeInformationService* via *HSCInformationServiceEP* endpoint
7. The response (HS_Code) from *HS_CodeInformationService* is forwarded to *TRRequest* sequence for mediation
8. *TRRequest* forwards the message to *TaxRateInformationService* via *TRInformationServiceEP* endpoint
9. The response (five tax rates) from *TaxRateInformationService* is forwarded to *PTARequest* sequence for mediation
10. *PTARequest* forwards the message to *PayableTaxAmountInformationService* via *PTRInformationServiceEP* endpoint
11. The message is received by the outSequence of the *IVDProxyService* Proxy service which performs the final mediation and steps and send the response to the service consumer (end user)

5.2 Implementation

WSO2 ESB configuration and creation of artifacts can be done either using WSO2 ESB Management Console or WSO2 Developer Studio. WSO2 Developer Studio provides a featured graphical mediation flow composer to create configurations for WSO2 ESB. For implementing the scenario described on the design section of this document WSO2 Developer Studio is used. WSO2 Developer Studio is the integrated development environment provided by WSO2 to develop SOA projects. It supports developing projects for most of the products in WSO2 product stack including ESB, AS, DSS and so on.

WSO2 Developer Studio is Eclipse based SOA development environment with graphical editor for creating ESB configuration. It provides a complete Eclipse based SOA development environment and immensely simplifies creation of artifacts with graphical editors and management of the links and dependencies between these services. It is possible to easily test, debug, and deploy artifacts as Composite Application aRchives onto servers such as the Application Server, Enterprise Service Bus, and Business Process Server. The

ESB graphical editor supports almost all building blocks of the WSO2 ESB including Proxy Service, REST APIs, Sequences, Endpoints and many more [41].

As shown in *Figure 5.4* above, the Imported Vehicle Declaration ESB contains three basic artifacts. These are endpoints, sequences and proxy service. The brief description of these ESB artifacts and the list of mediators and the ESB components that are included in the ESB implementation are described below in the following subsections.

5.2.1 Endpoints

An end-point is a logical abstraction over an external destination where WSO2 ESB has to deliver the message. The end point defined in WSO2 ESB can also take care of quality of service aspects like security and reliability corresponding to the external destination [2].

An endpoint may be specified as an address endpoint, WSDL based endpoint, a load balancing endpoint, a fail-over endpoint or an HTTP endpoint [38]. For this project an address endpoint is used. Address endpoint is an endpoint defined by specifying the EPR (Endpoint Reference) and other attributes of the configuration.

The Syntax for configuring the address endpoint is as follows.

```
<address uri="endpoint address" [format="soap11|soap12|pox|rest|get"] [optimize="mtom|swa"]
   [encoding="charset encoding"]
   [statistics="enable|disable"] [trace="enable|disable"]>
   <enableSec [policy="key"]/>?
   <enableAddressing [version="final|submission"] [separateListener="true|false"]/>?

   <timeout>
      <duration>timeout duration in milliseconds</duration>
      <responseAction>discard|fault</responseAction>
   </timeout>?

   <markForSuspension>
      [<errorCodes>xxx,yyy</errorCodes>]
      <retriesBeforeSuspension>m</retriesBeforeSuspension>
      <retryDelay>d</retryDelay>
   </markForSuspension>

   <suspendOnFailure>
      [<errorCodes>xxx,yyy</errorCodes>]
      <initialDuration>n</initialDuration>
      <progressionFactor>r</progressionFactor>
      <maximumDuration>l</maximumDuration>
   </suspendOnFailure>
</address>
```

The ‘uri’ and ‘format’ attributes specify the ERP of the target endpoint and the message format for the endpoint respectively.

In the implemented ESB there are five address endpoints that define the web services to be integrated through the ESB. The defined endpoints are the following:

BIPInformationServiceEP Endpoint

BIPInformationServiceEP Endpoint is an address endpoint that defines the application service *BankImportPermitInformationService* which is responsible for retrieving the bank import permit status of imported vehicle declaration. The endpoint is defined as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="BIPInformationServiceEP">
   <address uri="http://localhost:9764/services/BankImportPermitInformationService/"/>
</endpoint>
```

TAIPInformationServiceEP Endpoint

TAIPInformationServiceEP Endpoint is an address endpoint that defines the application service *TransportAuthorityImportPermitInformationService* which is responsible for retrieving the transport authority vehicle import permit status of imported vehicle declaration. *TAIPInformationServiceEP* endpoint is defined as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="TAIPInformationServiceEP">
   <address
uri="http://localhost:9764/services/TransportAuthorityImportPermitInformationService/"/>
</endpoint>
```

HSCInformationServiceEP Endpoint

HSCInformationServiceEP Endpoint is an address endpoint that defines the application service *HS_CodeInformationService* which is responsible for retrieving the HS_Code of the imported vehicle. *HSCInformationServiceEP* endpoint is defined as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="HSCInformationServiceEP">
   <address uri="http://localhost:9764/services/HS_CodeInformationService/"/>
```

```
</endpoint>
```

TRInformationServiceEP Endpoint

TRInformationServiceEP Endpoint is an address endpoint that defines the application service *TaxRateInformationService* which is responsible for retrieving the five tax rates for the imported vehicle. The endpoint is defined as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="TRInformationServiceEP">
   <address uri="http://localhost:9764/services/TaxRateInformationService/"/>
</endpoint>
```

PTAInformationServiceEP Endpoint

PTAInformationServiceEP Endpoint is an address endpoint that defines the application service *PayableTaxAmountInformationService* which is responsible for retrieving the total payable tax amount for the imported vehicle to be released. *PTAInformationServiceEP* endpoint is defined as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint xmlns="http://ws.apache.org/ns/synapse" name="PTAInformationServiceEP">
   <address uri="http://localhost:9764/services/PayableTaxAmountInformationService/"/>
</endpoint>
```

5.2.2 Sequences

A sequence is a logical group of a set of mediators arranged sequentially. It is used to define a sequence of mediators that can be invoked later by name. A given sequence may have an arbitrary number of mediators and message flows through them in sequential manner. A sequence can be either in-line or named. In-line sequences are sequences defined in the place they are used. Named sequences are sequences defined and saved with a name and can be recalled at different places by name. In other words, a named sequence can be reused in different places. Also note that a given sequence can call another sequence during its execution via the Sequence mediator [2]. In this ESB based integration solution, there are four named sequences defined. The sequences are described hereunder.

BIPRequest Sequence

BIPRequest Sequence is a set of mediators that are responsible for processing the Transport authority vehicle import permit status for the client that requested for vehicle declaration and forward the process to the *BIPInformationServiceEP* endpoint. The *BIPRequest* Sequence graphical view that shows the mediators involved is shown in *Figure 5.5*.

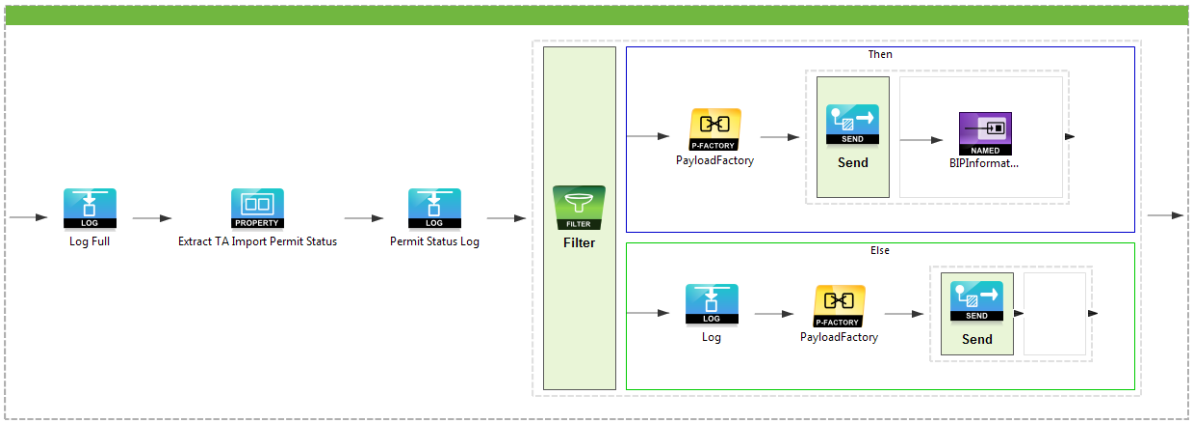

Figure 5.5: BIPRequest Sequence

The objective of this sequence is to extract the Transport Authority vehicle import permit status from the *TransportAuthorityImportPermitInformationService* response and construct the required payload suitable for *BankImportPermitInformationService*. This can be achieved via a property mediator (to extract the TA Import permit Status) and payload factory mediator.

As shown in the *Figure 5.5*, fist the log mediator is used to log the full response that we get from *TransportAuthorityImportPermitInformationService*. Then the TA Import Permit Status will be extracted from the response by using property mediator and the extracted value will be logged. Then by using the Filter mediator the TA Import Permit Status value will be checked and if the status is 'Granted' the payload will be constructed using a payload factory mediator and the constructed message will be sent to *BIPInformationServiceEP* endpoint by the Send mediator and the receiver for the response from *BankImportPermitInformationService* will be set to be *HSCRequest* sequence. If the TA Import Permit Status value is 'Denied' the message is logged and a message that indicates failure in TA Import Permit will be constructed using Payload factory and it will be sent to the outSequence of the proxy service.

HSCRequest Sequence

HSCRequest Sequence is a set of mediators that are responsible for processing the Bank import permit status for the client that requested for vehicle declaration and forwards the process to the *HSCInformationServiceEP* endpoint. The *HSCRequest* Sequence graphical view that shows the mediators involved is as shown in *Figure 5.6*.

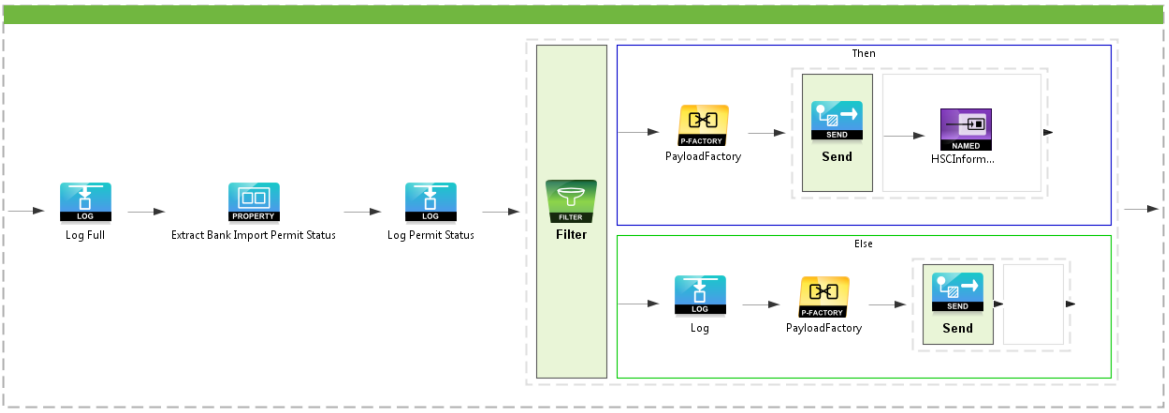

Figure 5.6: HSCRequest Sequence

The process flow in the *HSCRequest* sequence starts with logging the full response from *BankImportPermitInformationService* and then Bank Import Permit Status is extracted from the response by using property mediator and the extracted value will be logged. The extracted value will be evaluated by the filter mediator and either of the two options will be executed. If the value is 'Granted' the payload will be constructed by payload factory mediator and the constructed message will be sent to *HSCInformationServiceEP* endpoint by the Send mediator and the receiver for the response from *HS_CodeInformationService* will be set to be *TRRequest* sequence. If the Bank Import Permit Status value is 'Denied' the message is logged and a message that indicates failure in Bank Import Permit will be constructed by Payload factory and it will be sent to the outSequence of the proxy service.

TRRequest Sequence

TRRequest Sequence is a set of mediators that are responsible for processing the HS_Code for the vehicle declaration and forwards the appropriate request message to the *TRInformationServiceEP* endpoint to retrieve the tax rated to be paid. The *TRRequest* Sequence graphical view that shows the mediators involved is as shown *Figure 5.7*.

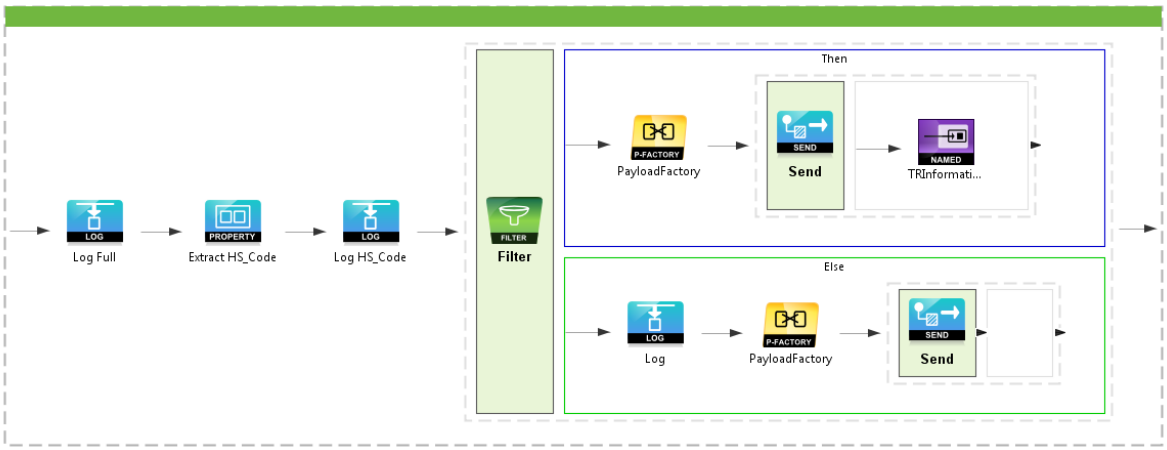

Figure 5.7: TRRequest Sequence

As shown in the *Figure 5.7*, *TRRequest* sequence begins by logging the full response message from *HS_CodeInformationService* and the HS_Code value is extracted from the response by the property mediator. The extracted HS_Code value will be evaluated by the filter mediator and either of the two options will be executed. If the HS_Code value is valid the payload will be constructed by payload factory mediator and the constructed message will be sent to *TRInformationServiceEP* endpoint by the Send mediator and the receiver for the response from *TaxRateInformationService* will be set to be *PTARequest* sequence. If HS_Code value is 'NOT FOUND' the message is logged and a message that indicates failure in retrieving the HS_Code will be constructed by Payload factory and it will be sent to outSequence of the proxy service.

PTARequest Sequence

PTARequest Sequence is a set of mediators that are responsible for processing the tax rates for the vehicle declared and forwards the formatted request message by the payload factory mediator to the *PTAInformationServiceEP* endpoint to retrieve the total payable tax amount to be paid. The *PTARequest* Sequence graphical view is as shown in *Figure 5.8*.

Figure 5.8: PTARequest Sequence

The PTARequest sequence starts by logging the full response from *TaxRateInformationService* and followed by five properties that extract the five tax rates i.e. Duty rate, Excise rate, VAT, Sur rate and Withholding rate. The extracted tax rates will be logged by using the log mediator and a payload that contains the five tax rates and the vehicle cost information will be constructed by a payload factory and it will be sent to *PTAInformationServiceEP* endpoint by the Send mediator and the receiver for the response from *PayableTaxAmountInformationService* will be *IVDProxyService* proxy service's 'outSequence'.

5.2.3 Proxy Service

A proxy service is a virtual service that receives messages and optionally processes them before forwarding them to a service at a given endpoint. It is a virtual service hosted in the ESB runtime and it allows performing necessary transformations and introduces additional functionality without changing existing service [38]. The proxy service mediates any accepted requests and forwards them to a specified endpoint, most of the time to an actual Web Service. The responses coming back from the target endpoint are mediated back to the client that sent the original service request. Proxy services often make references to sequences, endpoints and local entries.

There are different types of proxy services on WSO2 ESB namely Pass Through Proxy, Secure Proxy, WSDL Based Proxy, Logging Proxy, Transformer Proxy and Custom Proxy. For this demonstration Customs Proxy is used. In custom proxy service it is possible to customize the sequences, endpoints, transports, and other QoS settings.

In WSO2 ESB, a proxy service is built with an In-Sequence, an Out-Sequence, and a fault sequence.

In-Sequence

A request message coming in to a given proxy service will hit the In-Sequence defined for that proxy service. An 'inSequence' in a proxy service would decide how the message would be handled after the proxy service receives the message.

Out-Sequence

A response message coming from a concrete or a business service will go through the Out-Sequence defined for the corresponding proxy service. The 'outSequence' defines how the response is handled before it is sent back to the client. Unlike sequences and endpoints which can be stored and loaded from the registry, proxy services cannot be loaded from the registry. However a proxy service can make references to sequences and endpoints stored in the registry.

Fault sequence

It is also possible to associate a Fault-Sequence with a proxy service and it will get executed when an exception happens in a proxy operation. This sequence won't get executed for the exceptions thrown from the backend business services. Those will still go through the Out-Sequence.

IVDProxyService Proxy Service

In the implemented enterprise service bus a custom proxy service, *IVDProxyService*, is defined. As shown in *Figure 5.9*, the *IVDProxyService* has two sequences 'inSequence' and 'outSequence' that contains various mediators. The full request message coming to the proxy service will be logged by using the log mediator in the 'inSequence' first and then the values of the request message i.e. vehicle description, business license, cost, fright, insurance, other costs and amount will be extracted by using a property mediators. Then the vehicle description will be logged by using a custom log mediator and a payload will be constructed with the vehicle description and business license by PayloadFactory mediator. After the message is constructed it will be sent to *TAIPInformationServiceEP* endpoint by Send mediator and the receiver for the response from *TransportAuthorityImportPermitInformationService* will be set to be *BIPRequest*. When the response message arrives to the 'outSequence', it will be logged by a full log mediator and the response will be sent back to the requesting client by using the send mediator.

Figure 5.9: IVDProxyService Proxy Service

The *IVDProxyService* proxy service is a secured proxy service that can only be accessed by the legitimate users. It is only accessible by the group of users with admin privilege group. The green lock icon in *Figure 5.10* indicates the proxy service is secured and it can be accessed only by legitimate users.

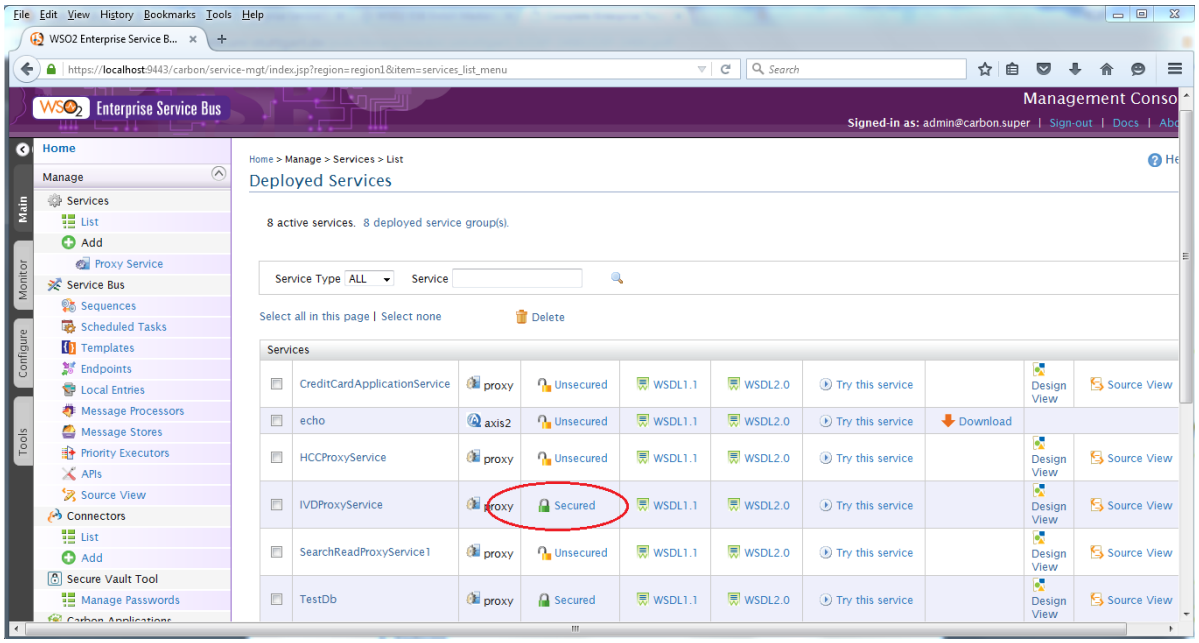

Figure 5.10: Secure IVDProxyService Proxy Service from the Management Console

5.2.4 Mediator

Mediator is the smallest functional unit in WSO2 ESB. A mediator is granular enough to perform a given specific task. WSO2 ESB comes with a rich collection of mediators addressing most of the common integration problems [2]. For this scenario different set of mediators are used for different purposes. The list of different mediators used and their description is described in the following sections.

Log Mediator

The Log mediator is used to log mediated messages [38]. The Log mediator can be used to log any incoming/outgoing messages. In the demonstration there are two types of log mediators used frequently: full and custom mediator. In custom log only the properties specified to the log mediator are logged. In full mediator all attributes included in the log level and any properties are logged.

The Log mediator configuration takes the following general form.

```
<log [level="string"] [separator="string"]>
  <property name="string" (value="literal" | expression="[XPath|json-eval(JSON Path)]")/>*
</log>
```

The 'level' attribute is used to specify how much information should be logged by the log mediator. This attribute can take values Simple, Headers, Full or Custom. The separator for properties and attributes logged can be configured using the 'separator' attribute.

Property Mediator

Properties provide the means of accessing various types of information regarding a message that passes through the ESB. Properties can be used to control the behavior of the ESB on a given message [38]. The property mediator is used to set or remove properties. Properties provide a way of accessing different attributes of the message, passing through the WSO2 ESB. It is also possible to use properties to change the behavior of the message flow as well as to set different attributes on the message passing through [2].

The syntax for configuring the property mediator is as follows.

```
<property name="string" [action=set|remove] [type="string"]
(value="literal" | expression="xpath") [scope=default|transport|axis2|axis2-client]
[pattern="regex" [group="integer"]]>
   <xml-element/>?
</property>
```

The 'name' attribute specifies the name of the property which needs to be either set or removed while the 'action' attribute specifies the exact action that needs to be carried out by the mediator. The 'value' attribute can be used to set a constant as the property value whereas the 'expression' attribute can be used to specify an XPath expression.

Send Mediator

The Send Mediator is used to send messages out to an endpoint. The Send Mediator also copies any message context properties from the current message context to the reply message received on the execution of the send operation, so that the response could be correlated back to the request [38].

The send mediator is configured using the following XML syntax.

```
<send [receive="string">
   (endpointref | endpoint) ?
</send>
```

An optional receiving sequence can be configured using the 'receive' attribute.

Filter Mediator

The Filter Mediator can be used for filtering messages based on an XPath, JSONPath or a regular expression. If the test succeeds, the Filter mediator executes the other mediators enclosed in the sequence. The Filter Mediator closely resembles the "If-else" control structure [38].

The configuration of the filter mediator takes the following form.

```
<filter (source="[XPath|json-eval(JSONPath)]" regex="string") | xpath="[XPath|json-
eval(JSONPath)]">
  <then [sequence="string"]>
    mediator+
  </then>
  <else [sequence="string"]>
    mediator+
  </else>
</filter>
```

The filter mediator either tests the given XPath expression as a boolean expression, or matches the result of the source XPath expression as a string against the given regular expression. If the condition evaluates to true, the filter mediator enclosed by the 'then' element will be executed. Failed messages will be mediated through the mediators enclosed by the 'else' element. Failed messages will be mediated through the mediators enclosed by the 'else' element.

PayloadFactory Mediator

The PayloadFactory Mediator transforms or replaces the contents of a message. Payload factory mediator is used to transform the message in desired output. WSO2 Payload Factory Mediator is for transforming or replacing message content in between the client and the backend server. Incoming message from client can be transformed by the Payload Factory mediator into the format expected by the service. Then the response message can be formatted into which is expected by the client.

Each argument in the mediator configuration could be a static value or an XPath expression. When an expression is used, argument value is fetched at runtime by evaluating the provided XPath expression against the existing SOAP message. Also the format of the request or response message can be configured which is mapped with the arguments provided in order.

The following syntax is used to configure the transformation performed by PayloadFactory mediator.

```
<payloadFactory media-type="xml | json">
   <format>"xmlstring"</format>
   <args>
      <arg (value="string" | expression=" {xpath} | {json} ")/>*
   </args>
</payloadFactory>
```

'format' sub-element of the mediator configuration specifies the format of the new payload. All $n occurrences in the format will be replaced by the value of the n th argument at runtime.

5.2.5 Application Services

There are five separate services running at different places in the scenario demonstrated. The application services for this demonstration are developed and deployed on WSO2 AS. The list of the services and their functionalities are described hereunder.

TransportAuthorityImportPermitInformationService

This service is capable of mapping the importer information and the imported vehicle information with the status of its Transport Authority import permit. It will process the given customers and the vehicles information and provide the information whether the customer is

granted a vehicle import permit for the vehicle to be imported or denied. The service is provided by Transport Authority. *TransportAuthorityImportPermitInformationService* application service is developed by using Eclipse Java EE IDE for Web Developers and is hosted on WSO2 AS application server. The service uses Excel database table as a back end to access and acquire the required information on the Transport Authority import permit information.

BankImportPermitInformationService

This service is capable of mapping the importer information and the imported vehicle information with the status of its bank import permit. It will process the given information about the Transport Authority vehicle import permit, importer customer and the imported vehicle and provide the information whether the customer is granted a bank import permit for the vehicle imported or denied. The service is provided by the Bank. The *BankImportPermitInformationService* application service is developed by using Eclipse Java EE IDE for Web Developers and is hosted on WSO2 AS application server. The service uses MySQL database as a back end to access and acquire the required information on the Bank import permit information.

HS_CodeInformationService

This service takes a description of the vehicle imported from the customer and retrieves the corresponding Harmonized Code for the imported vehicle described. This code is a unique identity for imported goods. It is important for retrieving the corresponding tax rates to be paid. The service is provided by ERCA. *HS_CodeInformationService* application service is developed by using Eclipse Java EE IDE for Web Developers and is hosted on WSO2 AS application server. The service uses Excel database table as a back end to access and acquire the required HS_Code information.

TaxRateInformationService

This application service is responsible for retrieving the list of type of taxes to pay for the imported product and the tax rate amounts. It searches from the Ethiopian_import_tariff_rate database. This Application service takes the HS_Code of the vehicle imported as an input and retrieves the five tax rates, Duty rate, Excise rate, VAT, Sur rate and Withholding rate,

for the vehicle imported. This service is provided by ERCA. *TaxRateInformationService* application service is developed by using Eclipse Java EE IDE for Web Developers and is hosted on WSO2 AS application server. The service uses Excel database table as a back end to acquire the tax rates value.

PayableTaxAmountInformationService

This Application service takes the five different tax rates for the vehicle imported i.e. customs duty, excise tax, VAT, surtax and withholding tax, cost of the vehicle, fright paid for the vehicle, insurance amount, other cost and number of the vehicle imported as an input and calculate the total payable tax amount for the vehicle to be released. This service is provided by ERCA. The *PayableTaxAmountInformationService* application service is developed by using Eclipse Java EE IDE for Web Developers and is hosted on WSO2 AS application server.

The implementation code for all the ESB component described above is attached at the end of this document.

5.3 Tools used for Development and Testing

For developing and testing the application services and the enterprise service bus based integration, the following tools are used.

- **WSO2 Developer Studio Version 3.7.1:** WSO2 Developer Studio is a complete tooling platform for easily developing, deploying, testing and debugging SOA applications. Developer Studio works in the popular open-source integrated development environment (IDE) Eclipse. For this project it is used for developing the ESB artifacts that can be deployed on the WSO2 ESB.
- **Eclipse Java EE IDE for Web Developers Version Kepler Service Release 2:** Eclipse Java EE IDE is used to develop the simulated applications.
- **WSO2 ESB Version 4.8.1:** WSO2 Enterprise Service Bus (WSO2 ESB) is used to deploy and configure the ESB Scenario.
- **WSO2 AS Version 5.2.1:** WSO2 Application Server (WSO2 AS) is used for deploying the application simulated so that the enterprise service bus can tested.

- **SoapUI Version 5.2.1:** SoapUI is used for testing the functionality of the ESB and the applications developed and deployed on WSO2 ESB and WSO2 AS.

5.4 Testing

For testing the implemented enterprise service bus that integrates the five different applications the SoapUI and TryIt is used. The screenshot of the ESB using these tools is as shown in the figures hereunder.

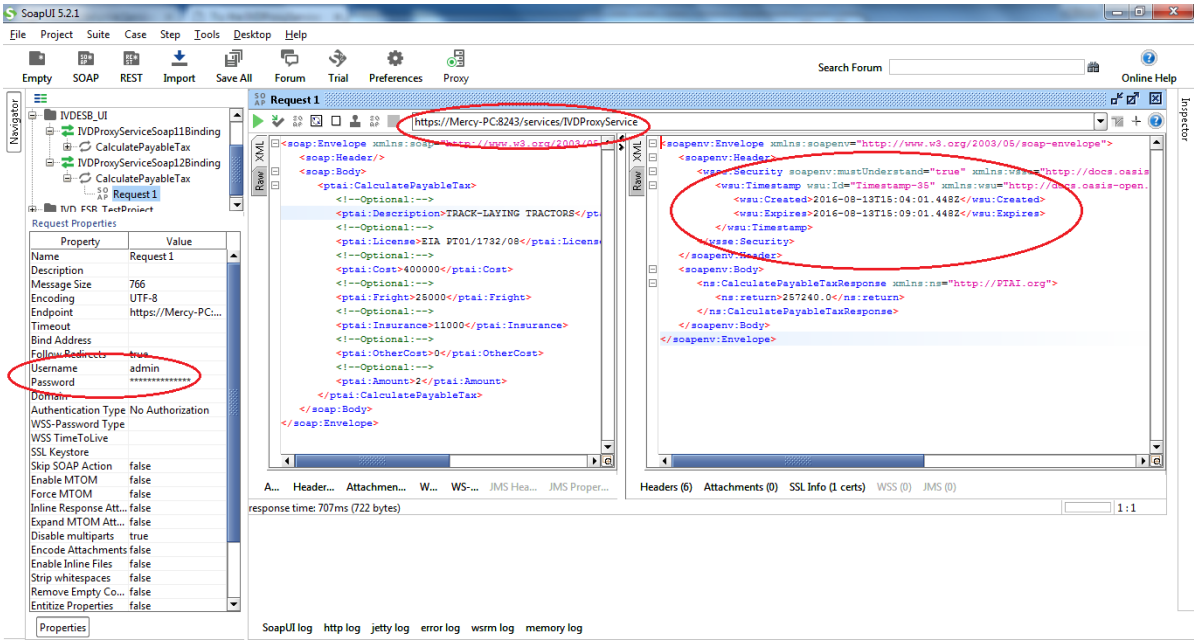

Figure 5.11: ESB test result using SoapUI1

Figure 5.11 shows the test result for the IVDProxyService proxy service by providing a legitimate user name and password as well as valid request input values and the response which indicates a secured communication with a valid payable tax amount for the inserted input information by using SoupUI.

The test result in *Figure 5.12* shows a valid request message sent with appropriate authentication account and the result indicates the transport authority vehicle import permit is denied by using SoupUI testing tool.

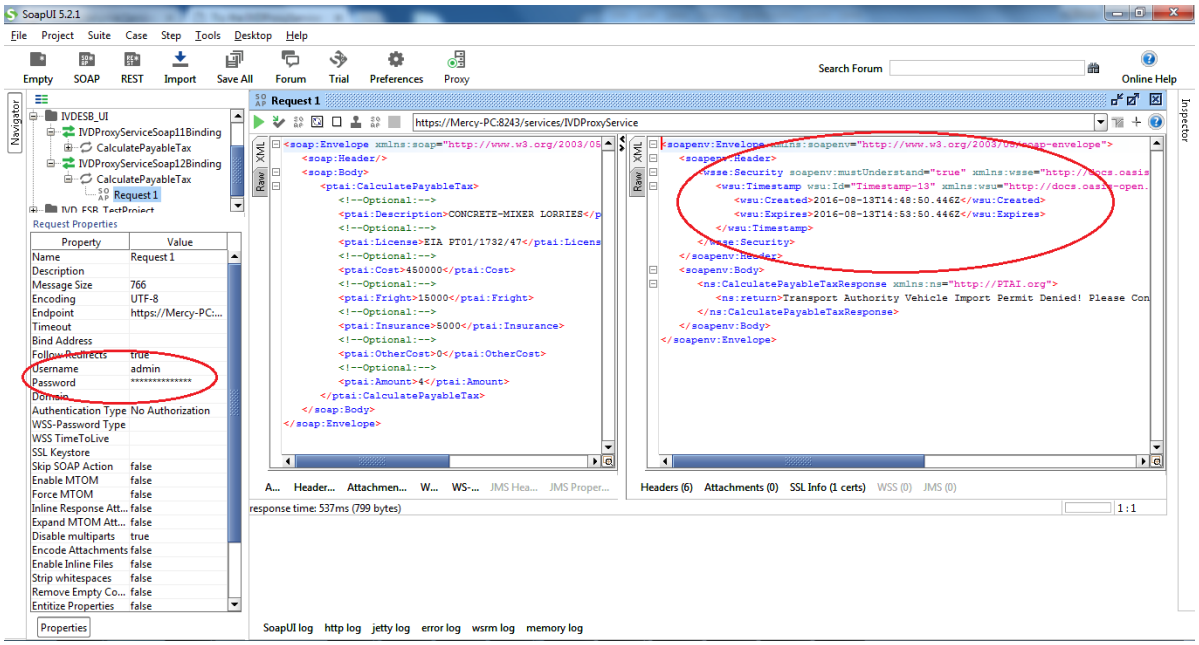

Figure 5.12: ESB test result using SoapUI2

The test result in *Figure 5.13* shows the failure due to accessing the ESB without providing legitimate user name and password by using SoapUI.

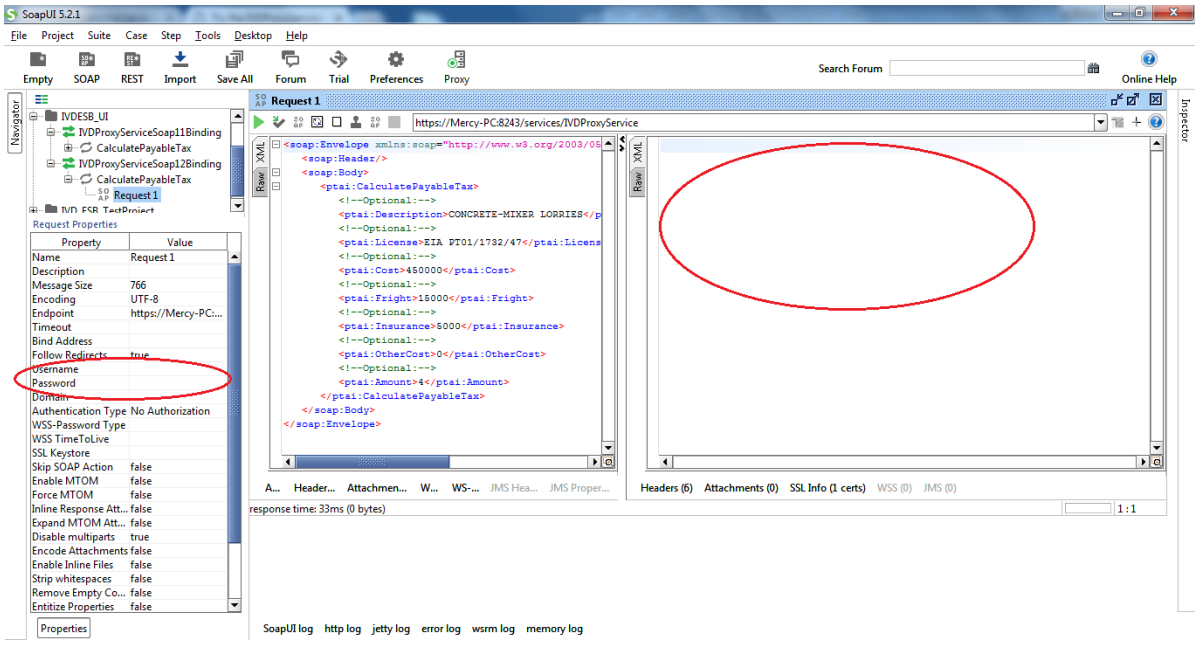

Figure 5.13: ESB test result using SoapUI3

Figure 5.14 shows test result by using TryIt testing tool providing a valid account and request information which provides a valid result that indicate a corresponding payable tax amount for the request values inserted.

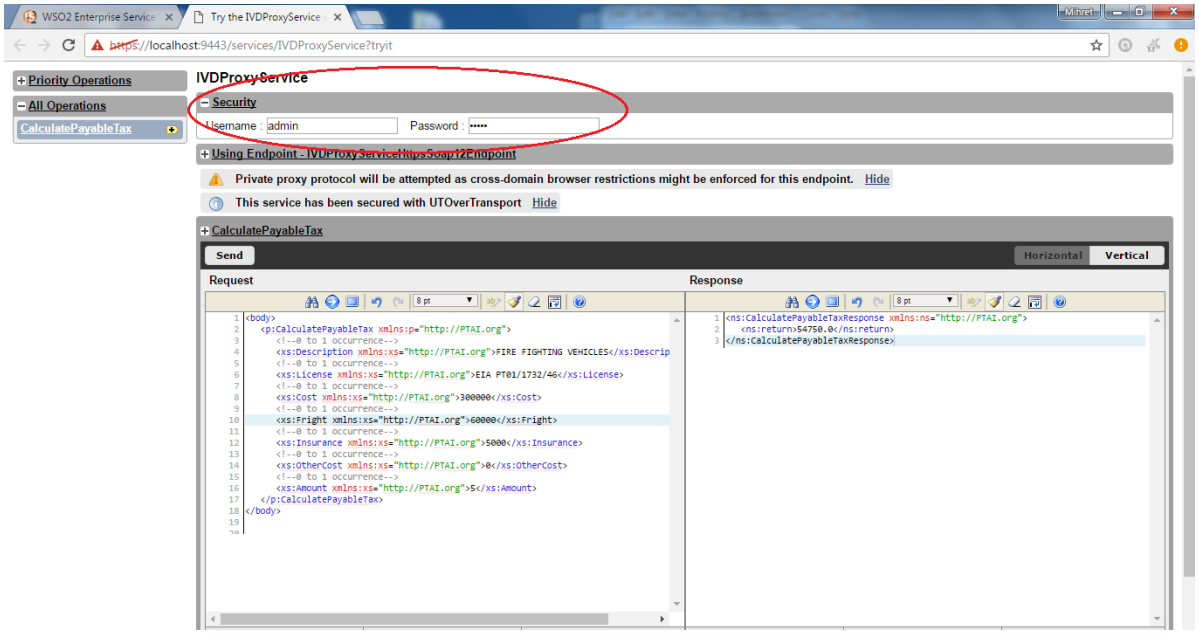

Figure 5.14: ESB test result using TryIt1

In *Figure 5.15*, the ESB is accessed by a legitimate user with a valid request value by TryIt testing tool and the result indicates bank import Permit is denied by displaying a message.

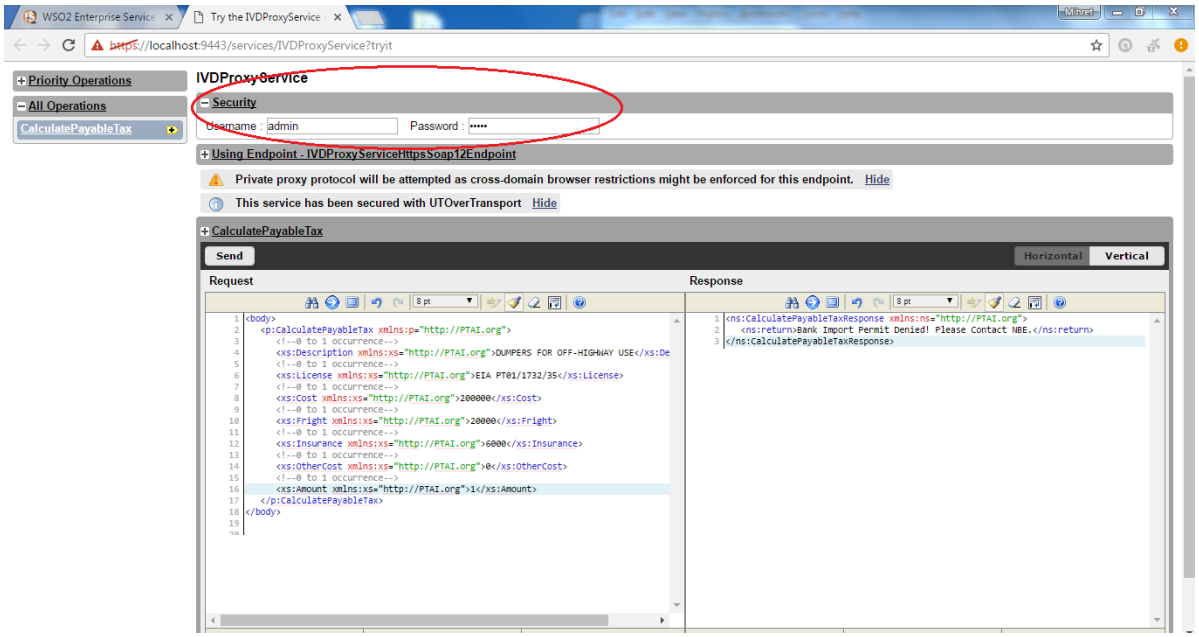

Figure 5.15: ESB test result using TryIt2

Figure 5.16 indicates the failure occurs by trying to access the ESB without providing legitimate account information by using a TryIt testing tool.

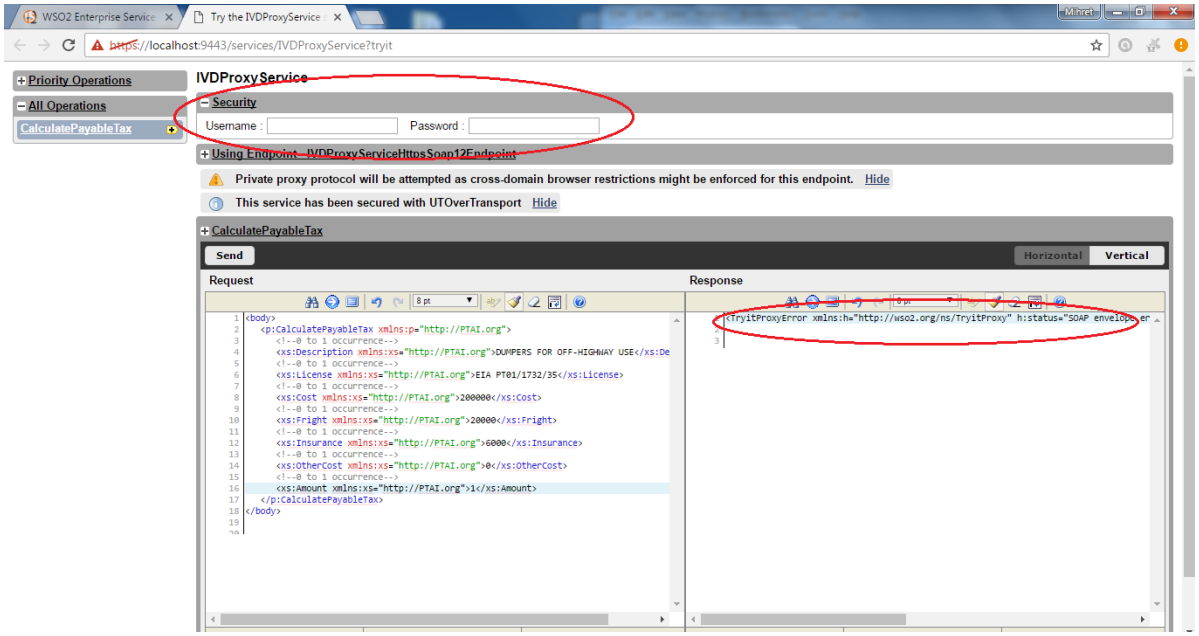

Figure 5.16: ESB test result using TryIt3

5.4.1 Functional Testing and Result Evaluation

The features and operational behavior of the ESB scenario demonstrated tested to ensure its correspondence with the specifications by using a functional testing. The goal of this testing is to verify whether the ESB developed meets the basic functionality of an ESB described in section *2.2.1* in Chapter Two of this document. It also aims to test the functionality of each of the applications to be integrated separately and also test whether the overall system functions as it should be.

For the functional testing, a functional testing document is prepared to be filled by different individuals. The document is divided in to nine test sets: Component Testing, Routing, Message Transformation, Message enhancement, Message processing, Service Orchestration, Database, System and Security test sets.

The proper functionality of each application services hosted on the application server is tested on the Component Testing test set. The implementation of the major capabilities of ESB are tested on Routing, Message Transformation, Message enhancement, Message processing, Service Orchestration and Security test sets and the overall developed ESB functionality as a system is tested on the System test set category of the functional testing document. The proper coordination of the backend databases with the application services and the ESB is tested on the Database test case section of the document. The functional testing document is attached at the *Annex B* section at the back of this paper.

For the functional testing five different individuals from different work environment are participated. The testers are: Customer Service Officer Biruk H/Mariam from CBE, Vehicle Importer Abreham Bersisa, G. Manager Soresa Tesfaye from SOWMECH Engineering Plc, Project Manager Tekachew Gobena from INSA and Software program coordinator Niguse Seid from ERCA. As observed from the functional testing document filled by the individuals listed above, all the functional requirements listed are fully functional. The result indicates all the five applications services are fully functional separately, all the incoming requests to the ESB are routed to the corresponding services properly and appropriate responses are routed back to the requester, all the required message structure transformations, message enhancements and message processing are handled properly by the ESB, the services coordinate to handle requests properly, retrieval of data from the Excel

and MySQL databases works as expected, the necessary security measures such as authentication of the ESB users, verification of users roles and responsibilities, delivery of messages through a secure communication channel and the password protectedness of the proxy service of the ESB during being accessed by external means are implemented successfully and the ESB system as a whole integrates the application services successfully.

Chapter 6

Conclusion and Future Work

An ESB is a service-oriented middleware solution that enables integration of heterogeneous systems by modeling application endpoints as services. It is an open source standard, message based, distributed integration infrastructure that provides routing, invocation and mediation services to facilitate the interactions of disparate distributed applications and services in a secure and reliable manner. It facilitates integration between disparate applications with different hardware and software platform.

As observed from the literature review and the scenario developed, an Enterprise Service Bus is an ideal solution for system integration. It is easy to develop without major change on existing systems; it is easily scalable; any maintenance on the ESB that might be required for the future is possible with less effort and cost; and it is not application dependent that means it can integrate any type of system Application without any preconditions. As mentioned on the comparison between proprietary and open source Enterprise Service Bus products, considering an open source ESBs for integration will benefit the organization from variety of angles. It is cost effective, modifications and customizations can be done without the request for license.

The integration solution described in this paper is ESB based integration that integrates the services provided by the different organizations to enable facilitated business process flow. This integration solution simplifies the procedures from the trader's perspective and increases the efficiency of operations within the agencies without the need for significant changes to internal processes and without relaxing the existing levels of control.

The Enterprise Service Bus will provide a means of integration between all the government entities that participate in customs import export procedures at ERCA. It facilitates the service procedures. From this integration the importers and exporters/Traders and also the organizations that participate in the process will benefit and the service will be facilitated.

ERCA is planning to establish an Electronic Single Window system. The system will be used to accelerate international trade by helping stakeholders access documents needed for customs clearance in one location. To successfully implement Electronic Single Window

system, it will require the different systems of the Governmental Organizations and Ministries that work together with ERCA to integrate. The institutions are National Bank of Ethiopia (NBE), Ministry of Trade (MoT), Ministry of Industry (MoI), Ministry of Health (MoH), Ministry of Communication & Technology (MoIT) and Ministry of Agriculture (MoA). Therefore, we recommend ERCA to consider open source Enterprise Service Bus integration approach for its Electronic Single Window project.

Future work could evaluate the currently available open source Enterprise Service Bus Products based on experiments in local environment and make the selection best suited for integrating Applications and Services in Ethiopian Organizations.

Reference

[1] D. Chappell, *Enterprise service bus*. Beijing: O'Reilly, 2004.

[2] P. Siriwardena, *Enterprise Integration with WSO2 ESB*. Birmingham: Packt Publishing, 2013.

[3] D. Dossot and J. D'Emic, *Mule in action*. Greenwich, Conn.: Manning, 2010.

[4] A. Grund, "Complete enterprise topologies with routing information of Enterprise Services Buses to enable cloud-migration", M.Sc. Thesis, University of Stuttgart, 2013.

[5] Z. Siddiqui, A. Abdullah, M. Khan and K. Alghathbar, "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*, vol. 6, no. 1, pp. 35-42, 2011.

[6] T. RahimSoomro and A. Hasnain Awan, "Challenges and Future of Enterprise Application Integration", *International Journal of Computer Applications*, vol. 42, no. 7, pp. 42-45, 2012.

[7] "Ethiopian Revenues and Customs Authority", *Erca.gov.et*, 2015. [Online]. Available: http://www.erca.gov.et. [Accessed: 12- Feb- 2015].

[8] J. Jenkov, "Enterprise Service Bus (ESB)", *Tutorials.jenkov.com*, 2014. [Online]. Available: http://tutorials.jenkov.com/soa/esb.html. [Accessed: 12- Feb- 2015].

[9] A. Goel, "Enterprise integration EAI vs. SOA vs. ESB", *Infosys Technologies White Paper*, vol. 87, 2006.

[10] D. Kus`ak, "Comparison of Enterprise Application Integration Platforms", M.Sc. Thesis, Charles University in Prague, Department of Software Engineering, 2010.

[11] K. Wähner, "Choosing the Right ESB for Your Integration Needs", *InfoQ*, 2013. [Online]. Available: https://www.infoq.com/articles/ESB-Integration. [Accessed: 12- Feb- 2015].

[12] T. Rademakers and J. Dirksen, *Open source ESBs in action*. Greenwich [Conn.]: Manning, 2009.

[13] A. Manes, "Enterprise Service Bus: A Definition", *Burton Group*, pp. 1-35, 2007.

[14] F. Menge, "Enterprise service bus", in *Free and open source software conference*, 2007, pp. 1-6.

[15] D. Le Sage, B. Long, D. Mison and T. Wells, *JBoss Enterprise SOA Platform 5 ESB*

Programmers Guide. Red Hat, Inc, 2013.

[16] X. Zhi, B. Jin and W. Yufei, "The Research and Implementation of Power Application System Integration Based on Enterprise Service Bus", 2010 *IEEE International Conference*, 2010, pp. 466-469.

[17] "eBay case study: eBay uses 100% open source WSO2 ESB to process more than 1 billion transactions per day", *http://wso2.com/*, 2011. [Online]. Available: http://wso2.com/download/getfile/wso2-ebay-case-study.pdf. [Accessed: 10- Mar- 2015].

[18] W. UTOMO and T. WELLEM, "The Implementation of Enterprise Service Bus (ESB) in Graduation Business Process Integration", *Journal of Theoretical & Applied Information Technology*, vol. 62, no. 2, pp. 364-370, 2014.

[19] L. Yang and L. Shi-Bin, "A Research of Remote Sensing Data Service System Based on Mule ESB", *2012 International Conference on Computer Science and Service System*, pp. 25-28, 2012.

[20] W. Jiang, S. Zheng, C. Wu and H. Pan, "Research on application of Web based ESB in School Common Data Platform", *Proceedings of 2009 4th International Conference on Computer Science & Education*, pp. 1311-1315, 2009.

[21] United Nations Centre for Trade Facilitation and Electronic Business (UNCEFACT), "Recommendation and Guidelines on Establishing a Single Window", New York, 2005.

[22] Eleni Araya, "New Single Window System to Ease International Trade Process", *Fortune*, 2013. [Online]. Available: http://addisfortune.net/articles/new-single-window-system-to-ease-international-trade-process. [Accessed: 24- Feb- 2015].

[23] Capital, "New Single Window System to Streamline International Trade", 2014. [Online]. Available: http://capitalethiopia.com/2014/06/23/new-single-window-system-to-streamline-international-trade/#.WBn3bC197IU. [Accessed: 24- Feb- 2015].

[24] K. Vollmer, M. Gilpin and S. Rose, "The Forrester Wave™: Enterprise Service Bus, Q2 2011", *Forrester Research*, 2011.

[25] "ESB comparison, How to choose a reliable and fast enterprise service bus", *www.yelno.com*, 2015. [Online]. Available: http://offer.yenlo.com/free-whitepaper-enterprise-service-bus-comparison. [Accessed: 22- Jun- 2015].

[26] S. Anfar, "[Article] WSO2 ESB Performance Round 7.5 | WSO2 Inc", *Wso2.com*, 2014. [Online]. Available: http://wso2.com/library/articles/2014/02/esb-performance-round-7.5/. [Accessed: 06- Sep- 2015].

[27] K. Röppänen, "Requirements for an Enterprise Application Integration Tool", M.Sc. Thesis, University of Tampere, 2013.

[28] R. Cope, "On-Demand Webinar: A Comparison of Open Source Software ESB Solutions", *Openlogic.com*, 2010. [Online]. Available: http://www.openlogic.com/events/on-demand-webinars/comparison-open-source-software-esb. [Accessed: 02- Feb- 2015].

[29] J. Thompson, Y. V. Natis, M. Pezzini, D. Sholler, R. Altman and K. Iijima, "Magic Quadrant for On-Premises Application Integration Suites", *Gartner*, 2013.

[30] S. Ahuja and A. Patel, "Enterprise Service Bus: A Performance Evaluation", *Communications and Network*, vol. 03, no. 03, pp. 133-140, 2011.

[31] "Welcome to Apache ServiceMix!", *Servicemix.apache.org*, 2015. [Online]. Available: http://servicemix.apache.org. [Accessed: 10- Jun- 2015].

[32] G. Behara, "Service Integration", *BPTrends*, 2008.

[33] J. Fenner, "Enterprise Application Integration Techniques", 2003.

[34] A. Zelenka, "Open Source ESBs for Application Integration (SOA Optional)", *RedMonk*, 2007.

[35] "Understanding Enterprise Application Integration - The Benefits of ESB for EAI", *MuleSoft*, 2014. [Online]. Available: https://www.mulesoft.com/resources/esb/enterprise-application-integration-eai-and-esb. [Accessed: 12- Feb- 2015].

[36] R. Woolley, "Enterprise Service Bus (ESB) Product Evaluation Comparisons" Department of Technology Services, Utah Department of Technology Services, Salt Lake City, 2016.

[37] R. Blaauboer, "A primer on Enterprise Service Bus (ESB) and WSO2", *Yenlo Knowledge Blog*, 2015.

[38] "WSO2 Enterprise Service Bus Documentation - Enterprise Service Bus 4.8.1 - WSO2 Documentation", *Docs.wso2.com*, 2014. [Online]. Available: https://docs.wso2.com/display/ESB481/WSO2+Enterprise+Service+Bus+-

Documentation. [Accessed: 11- Jun- 2015].

[39] "Understanding Integration From A "Needs-Based" Perspective - Mule vs. JBoss ESB", *MuleSoft*, 2016. [Online]. Available: https://www.mulesoft.com/resources/esb/understanding-integration-needs-based-perspective-mule-vs-jboss-esb. [Accessed: 10- Jun- 2015].

[40] J. Wu and X. Tao, "Research of enterprise application integration based-on ESB", *2010 2nd International Conference on Advanced Computer Control*, vol. 5, pp. 90-93, 2010.

[41] "WSO2 Developer Studio Documentation - Developer Studio 3.8.0 - WSO2 Documentation", *Docs.wso2.com*, 2016. [Online]. Available: https://docs.wso2.com/display/DVS380/WSO2+Developer+Studio+Documentation. [Accessed: 16- Apr- 2016].

[42] "Enterprise Integration Patterns with WSO2 ESB - Enterprise Integration Patterns with WSO2 ESB - WSO2 Documentation", *Docs.wso2.com*, 2016. [Online]. Available: https://docs.wso2.com/display/IntegrationPatterns/Enterprise+Integration+Patterns+-with+WSO2+ESB. [Accessed: 26- Apr- 2016].

[43] "Foreign Exchange Directives", *National Bank of Ethiopia*, 1998. [Online]. Available: http://www.nbe.gov.et/pdf/Consolidated%20Forex.pdf. [Accessed: 11- Jun- 2015].

[44] R. ZOTA, R. MOLEAVIN and L. CIOVICĂ, "Enterprise Service Bus - A Backbone for SOA", *Recent Researches in Business and Economics, Proceedings of the 4th WSEAS World Multiconference on Applied Economics, Business and Development*, pp. 241 - 244, 2012.

[45] D. Minoli, *Enterprise architecture A to Z*. Boca Raton: CRC Press, 2008.

[46] C. Thom and F. Jones, *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus*. 11g Release 1 (11.1.1.6.3), Oracle, 2012.

Annex A - Functionality Testing Document

In this testing the features and operational behavior of the ESB demonstrated will be tested to ensure it corresponds to its specifications. The goal of this testing is to verify whether the ESB developed meets the intended ESB functionalities mentioned in Chapter two of the document.

Project Name	**Open Source ESB Based Integration: The Case of ERCA**
Tested By	
Position	
Date	

Test Cases	**Procedure**	**Expected Result**	**Pass/ Fail**	**Actual Results/ Comments**
Component Testing				
Transport Authority Service Functionality	1. Navigate to the Transport Authority Import Permit Web service from the Application service 2. Click on TryIt 3. Insert input value	The appropriate status of the import permit displayed		
Bank Import Permit Service Functionality	1. Navigate to the Bank Import Permit Web service from the Application service 2. Click on TryIt 3. Insert input value	The appropriate status of the import permit displayed		
HS_Code Retrieval Service Functionality	1. Navigate to the HS_Code Retrieval Web service from the Application service 2. Click on TryIt 3. Insert input value	The appropriate HS_Code displayed		
Tax Rate Retrieval Service Functionality	1. Navigate to the Tax Rate Retrieval Web service from the Application service 2. Click on TryIt 3. Insert input value	The appropriate Tax Rates displayed		

Payable Tax Retrieval Service Functionality	1. Navigate to the Payable Tax Retrieval Web service from the Application service 2. Click on TryIt 3. Insert input value	The Payable Tax amount displayed		
Routing				
Incoming request from client reach to an appropriate web service through ESB	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button	The request is passed to the appropriate Web services and no error massage displayed		
Appropriate response for the request made by user reach to the user through the ESB	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button 4. Check on the response from the response window	The request is routed to the appropriate web services and the result is displayed with no error message		
Message Transformation				
The Appropriate message structure for each web services is provided by the ESB	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button 4. Check on the response from the response window 5. Check for any error or exception message displayed due to message structure format miss match	The request processed without any message structure format miss match error message		
Message Enhancement				
The appropriate message enhancement for the web services is handled by the ESB	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button 4. Check on the response from the response window 5. Check for any error or exception message displayed due to missed input value for services	The request processed without any to missed input value for web services error message		

Message Processing				
Manage state, accept input request and ensure delivery of results to	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button 4. Check on the response from the response window	The ESB manages states and process the request and result is displayed		
Service Orchestration				
Coordination between the Web Services takes place to fulfill a single service request	1. Open SoapUI 2. Insert the input value on the request window 3. Click on send button 4. Check on the validity of the response that coordinates the services	The ESB manages the coordination between the variety of services hosted and the appropriate result is displayed		
Database				
The data retrieved from the Excel table of the transport authority vehicle import permit status through the ESB is the same as the actual data on the Excel table	1. Navigate to the Transport Authority Import Permit Web service from the Application service 2. Click on TryIt 3. Insert input value 4. Click on send button 5. Cross check the status result displayed with the actual status on the excel table for the input value inserted	The status result displayed from the service is the same as the status on the table		
The data retrieved from the MySQL database table of the Bank authority import permit status through the ESB is the same as the actual data on the database table	1. Navigate to the Bank Import Permit Web service from the Application service 2. Click on TryIt 3. Insert input value 4. Click on send button 5. Cross check the status result displayed with the actual status on the MySQL database table for the input value inserted	The status result displayed from the service is the same as the status on the table		

The data retrieved from the excel table of the HS_Code retriever is the same as the actual data on the database table	1. Navigate to the HS_Code Retrieval Web service from the Application service 2. Click on TryIt 3. Insert input value 4. Click on send button 5. Cross check the status result displayed with the actual status on the excel table for the input value inserted	The status result displayed from the service is the same as the status on the table		
The data retrieved from the excel table of the Tax rate retriever service is the same as the actual data on the database table	1. Navigate to the Tax Rate Retrieval Web service from the Application service 2. Click on TryIt 3. Insert input value 4. Click on send button 5. Cross check the status result displayed with the actual status on the excel table for the input value inserted	The status result displayed from the service is the same as the status on the table		
System				
Handling the validity check for invalid data entry	1. Open SoapUI 2. Insert an invalid input value on the request window 3. Click on send button 4. Check the response returned	Appropriate message that describe the case displayed		
The final result displayed from the web services tested separately is the same as the result obtained from the ESB for the same input value	1. Navigate to the Web service from the Application service one by one and use TryIt 2. Open SoapUI and execute the ESB with the similar input value 3. Cross check the status result displayed from the ESB and the Web services executed	The final result obtained from the web services and from the ESB is the same		

Important operations on the ESB are written in log files and the information is traceable	1. Ruth the WSO2 ESB 2.Login to the ESB 3. Perform some tasks 4. Browse to the log file location and open and check the log entry	Appropriate information is written on log file and the activity is traceable		
Security				
ESB authenticate its users	1. Run the WSO2 ESB 2. On login page try to login without account 3. Try to log in with a valid Account	The WSO2 ESB rejects users request with invalid account and allows to login to the system with a valid account		
The ESB checks the privilege of the user account and provide the main window accordingly (Verify the user roles and their rights)	1. Run the WSO2 ESB 2. On login page try to login without account with user privilege 3. Check the main window provided is with user privilege 4. Try to log in with administrator privilege Account 5. Check the main window provided is with Admin privilege	The user main window is displayed for account for user and admin main window for administrator account is displayed (The user should not be able to access the admin page and admin privilege)		
The system locks out the account if the user enters wrong password several times	1. Run the WSO2 ESB 2. On login page try to login with wrong password several times	The system locks out the user		
The ESB system submits and receives data via secure web page (HTTPS Page)	1. Run the WSO2 ESB 2. Check the URL of the system on the browser	The URL of the page of the system is HTTPS		

If password is changed the user should not be able to login with the old password	1. Run the WSO2 ESB 2. Login to the ESB system 3. Change the password for the account 4. Log out 5. Login to the account with the old password	The system shouldn't be able to log the user in		
The proxy service should be secured	1. Run the WSO2 ESB 2. Login to the ESB system 3. Navigate to proxy services 4. Look for the IVDProxyService 5. Check its security status	The security status of IVDProxyService proxy service is secured with a green locked key icon		
The external interface applications that are used to access the ESB should be authenticated before using the ESB Service and provide the result	1. Open SoapUI 2. Insert input value on the request window 3. Insert the user name and password 4. Click on send button 5. Check the response returned	Error message should be displayed for accessing with wrong accounts and trying to send request without providing user name and password and should display result value for the request with correct account with the privilege to use the ESB		

Annex B - Interview Questions

Ethiopian Revenue and Customs Authority

1. What is the overall process flow for declaring an imported vehicle?
2. What is expected from the importer/trader/clearing agent to declare imported vehicle?
3. What are the preconditions that the importer should fulfill before starting imported vehicle declaration process?
4. What are the documents that are required for vehicle declaration process?
5. Is there an automated system in use for processing declaration process? If yes, briefly describe the system.
6. What are the governmental organizations that participate in the imported vehicle declaration process?
7. How is the collaboration between the different organizations work process handled considering the imported vehicle declaration process?
8. Is there any plan for integration between the varieties of governmental organizations that participate on customs import/export process?
9. It is on the news that ICF and the International Finance Corporation (IFC) are working with the Ethiopian Revenues and Customs Authority (ERCA) that aims to reduce the procedures, time and costs of international trade transactions in Ethiopia by deploying an end-to-end electronic Single Window System for trade. Can u explain the project briefly?
10. What is the status of the electronic Single Window project?
11. Is there any development plan reached to handle the integration between the participating governmental organizations?

Transport Authority

1. What is expected from an importer to import vehicles to Ethiopia?
2. To obtain a vehicle Import permit, what is expected from the importer to fulfill?
3. What are the preconditions that the importer should fulfill to obtain vehicle import permit?
4. Is there a list of vehicles model and type that are allowed to be imported to Ethiopia?

   If yes, is it possible to get document that describe the list

5. Is there a list of countries or source of origin that is possible to import from? If so, describe.
6. What kind of document is provided by the authority to the importer to confirm the import permit?
7. Is there any automated system that the organization uses for this process? If yes, describe the systems. If paper based, describe how the process is under gone and what kind of document s kept in the archive.
8. What kinds of documents are required from the importer to process the import permit?
9. In average how long will it take to process for an import permit?

10. Is there any payment for the process? If yes, how much and how is the payment taken care of?
11. Is there any vehicle standard inspection measure taken place? If so what is the measure and how?

Bank

1. What is expected from an importer to obtain bank import permit for imported vehicles?
2. What are the preconditions that the importer should fulfill to obtain bank import permit?
3. What kind of document is provided by the bank to the importer to confirm the bank import permit is granted?
4. Is there any automated system that the organization uses for this process? If yes, describe the systems. If paper based, describe how the process is under gone and what kind of document s kept in the archive.
5. What kinds of documents are required from the importer to process the bank import permit?
6. In average how long will it take to process for bank import permit?
7. How does the organization communicate with the Transport Authority to check the vehicle import permit status?
8. Is there any means of communication between Transport Authority, Bank and Ethiopian Revenue Customs Authority considering the imported vehicle declaration process? If yes, explain briefly. If no, how does the business flow by various organizations coordinate?

Importer/ Trader/ Clearing Agent

1. What is the overall process (business process) to import a vehicle?
2. What are the organizations that involve in the business process above?
3. Which organization is responsible for which business process?
4. Briefly describe the process step by step according to their occurrence for the imported vehicle declaration?
5. What are the required documents and preconditions required by the organizations?
   5.1. What are the documents required by Bank to obtain L/C and Bank import Permit?
   5.2. What are the documents required by Transport authority to obtain vehicle import permit?
   5.3. What are the documents required by ERCA to process imported vehicle declaration?
6. How long does it take to declare a single imported vehicle?

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____