UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty or Science and Technology

Department of Computer Science

Information Logistics Service Router

An ESB for integrating enterprise services

—

Marius Lundblad

INF-3981 Master thesis in Computer Science, June 2015

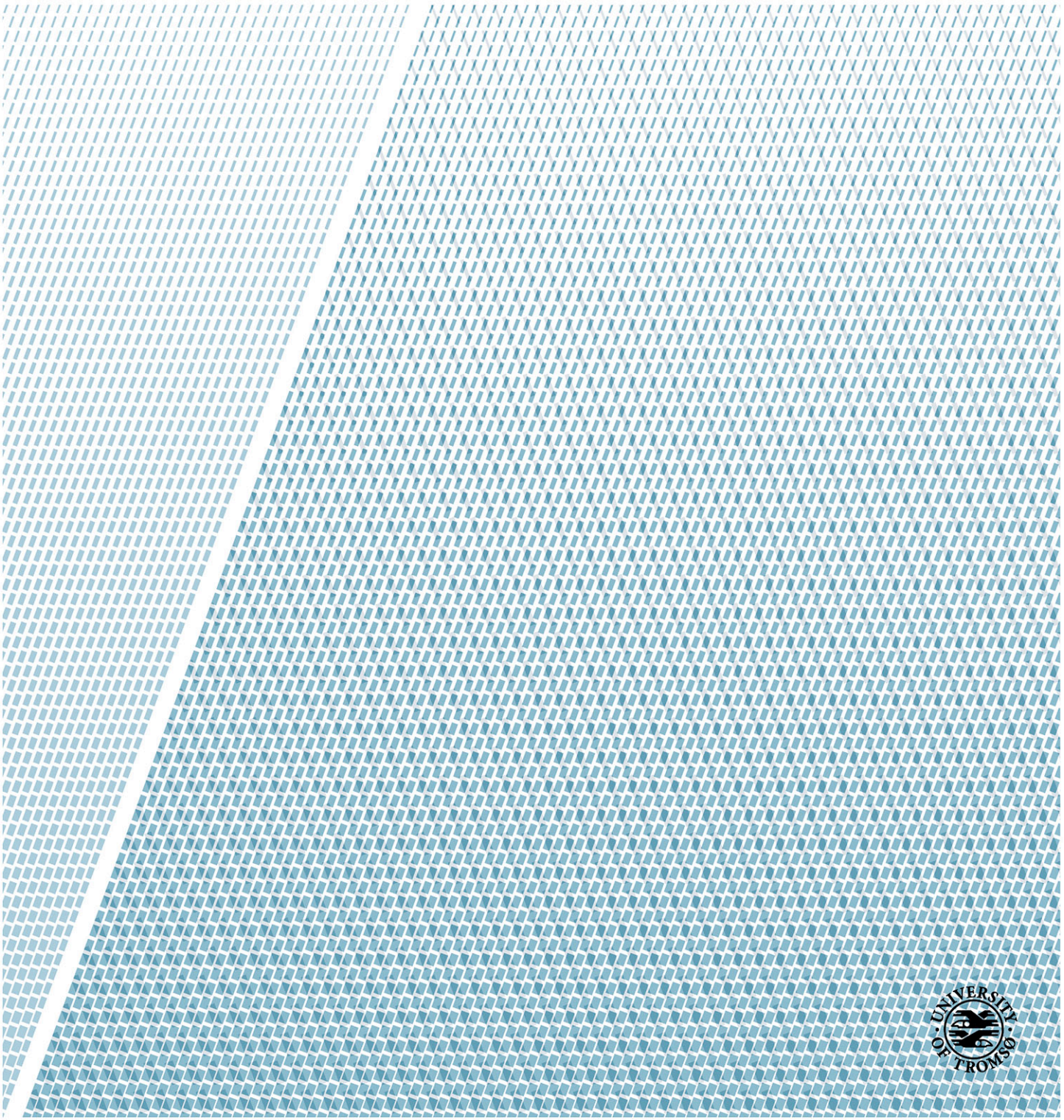

Table of Contents

List of Figures

List of Tables

ABSTRACT

Jupiter System Partner is currently working on a product idea called Information Logistics. The product focuses on collecting information via modules from several services in an enterprise system. The product will be based upon financial systems which can be customized via modules. Specifically, the product will be based upon their existing system called eSecretary which extends the economy system with a document storage where invoices, packaging documents and signatures can be stored.

One of the limitations addressed by the managers at Jupiter is the lack of an integration platform where the modules and implementation can be gathered. Part of the Information Logistics product idea is a web interface where customers which have issued invoices can check the progress of their orders. This is currently not possible as the eSecretary system is deployed in company domains. The managers have therefore posted a request for a platform which will publish functionality for checking order progress and other modules.

This thesis addresses the request and presents Information Logistics Service Router, an Enterprise Service Bus (ESB) which integrates the eSecretary web service and other services and applications into a common access point. Information Logistics Service Router is a BizTalk application which serves as a basis for the Information Logistics product idea. The system integrates applications and services into a common access point using the BizTalk platform.

The evaluation of Information Logistics Service Router shows that the system implements a simple, but useful integration basis for Information Logistics compared to other integration platforms such as Mule ESB and IBM WebSphere MQ.

Acknowledgements

I wish to express my sincere thanks to CEO at Jupiter System Partner, Kolbjørn Engeseth, for providing me with all the necessary facilities for the research in this thesis.

I extend my sincere thanks to my main supervisor, Anders Andersen, for the continuous encouragement, support, and invaluable guidance and expertise during the work on this master thesis.

I am also grateful to my coworker at Jupiter System Partner, Tony Liafjell. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement to me.

I take this opportunity to express gratitude to my fellow students at the faculty of Computer science and my friends for their support. I also thank my family for the invaluable encouragement, support and attention.

Last I want to place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand or supported me in this venture.

Chapter 1

Introduction

Web services are defined by the World Wide Web Consortium (W3C)[1] as software systems designed to communicate over a network in a machine-to-machine interaction. In 2003 the editor in chief for *InfoWorld*, Eric Knorr, predicted that 2004 would be the year of web services as a low-cost alternative to proprietary middleware [16].

In 2008, Eyhab Al-Masri and Qusay H. Mahmoud published a paper [19] which conducted a thorough analytical investigation on the plurality of web service interfaces that exist on the web today. The article concluded that a search engine was needed to collect information of the existing web service interfaces, since web service access points were no longer a scarce resource.

In 2010, Abdelghani Benharref, Mohamed Adel Serhani and Salah Bouktif presented a paper [20] which proposed a managerial community of web services that is able to monitor and certify the quality of web services in other communities. The article states that the number of web services will be growing exponentially for the next 5 years. Therefore the need for categorizing and classifying web services, might be crucial for the success of an underlying Service-Oriented Architecture (SOA). The article states that web service providers are trying to maximize their revenues by creating appropriate communities.

In 2014, Shakab Mokarizadeh, Peep Küngas and Mihhail Matskin presented a paper [21] which proposes a plausible solution for finding missing web services and measuring the added-value of the introduced services. The article was conducted due to the increasing presence and adoption of web services, and how it has promoted the significance of management of new service development for service developing sectors.

As stated by [19, 20, 21], companies are, and have been, working toward getting more data and business logic available for other services and applications over the network. This is normally achieved by creating some form of web service which implements functionality to collect and operate on data supplied by the customer. As described by these

[1] `http://www.w3.org/` (collected 14.05.15.)

articles, the number of web service access point are increasing drastically. The increase demands a unified way to offer web services, and prevent recurring business logic and data.

A web service contains an interface described in a machine-processable format, most often in the form of a Web Service Description Language (WSDL)[2]. Other systems interact with web services as instructed in the WSDL-file. The interaction between the two parts usually takes form of Simple Object Access Protocol (SOAP)[3] messages. The interaction is conveyed using HTTP[4] where operations are called using XML[5] serialization or in combination with other web standards. The rapid growth and usage of web services have made it troublesome for developers to manage existing interactions between applications and services.

Many companies deploy several services residing at their each individual end-point which can cause confusion when creating an interaction with an application. Also services have a tendency to create redundant service functionality residing at several service locations.

In this thesis we look into the possibility of lowering the complexity of several service end-points and management of existing applications by creating an Enterprise Service Bus (ESB)[6]. An ESB is an integration architecture which we will go closer into in chapter 3. By utilizing an ESB, the service end-point will always stay the same and access management to web services can be handled more intuitive. In this paper we will explore the possibility to create such a system in BizTalk Server 2013[7], which is a tool for integrating services and applications. Through the paper we will propose a prototype implementation of how an ESB can be implemented in a company setting. The paper will also look at other platforms such as Mule ESB and IBM WebSphere MQ to see how these perform compared to BizTalk. The platforms mentioned here will be explained and presented in chapter 3.

This thesis has been conducted in cooperation with Jupiter System Partner which has requested such a system, and has therefore been part of the evaluation of the prototype implementation. Jupiter System Partner is a company located in Tromsø, started by Kolbjørn Engeseth and Odd Halvard Bjørnstad which focuses on developing enterprise solutions for both the private and public sector. Through 12 years they have developed solutions such as eSecretary, UtleieSentral and Jupiter Innsyn.

The implemented system is built on Jupiter System Partner's requirements, and will be part of a project started in cooperation with *Innovasjon Norge*[8].

[2] `http://www.w3.org/TR/wsdl` (collected 28.04.15.)

[3] `http://www.w3.org/TR/soap/` (collected 28.04.15.)

[4] `http://www.webopedia.com/TERM/H/HTTP.html` (collected 28.04.15.)

[5] `http://www.w3.org/XML/` (collected 28.04.15.)

[6] `https://www.mulesoft.org/what-esb` (collected 10.03.15.)

[7] `https://www.microsoft.com/en-gb/server-cloud/products/biztalk/overview.aspx` (collected 10.03.15.)

[8] `http://www.innovasjonnorge.no` (collected 08/04/15)

1.1 Importance

Web services are used in almost every application that is developed these days. As described in [19, 20, 21], and stated by Anne Thomas Manes [17], "*almost every software vendor is building support for web services into its platforms, languages, and tools*". Due to recent development in how applications are being developed [18], programmers and companies choose to implement web services which provide crucial application data instead of integrating the service into a static application. This model of developing was introduced as part of the Service-Oriented Architecture (SOA)[9], which is an architecture that enforces reusability and decoupling of applications and services.

These architectures forwards the use of decoupled applications, where the business logic is mostly placed in web services which have no direct connection to the application itself.

Due to this new way of developing applications, companies are publishing and selling access to web services. The reasoning being to forward the progress and keep ahead of the competition in providing crucial data, as described by *JBoss Inc*'s article [18]. The constant increase of web services demands a unified way to provide several web services for clients. This has given birth to software models such as ESB and Remote Method Invocation (RMI)[10].

Several enterprises does not make use of these models since they have been trivial to implement and usually requires several web services to be beneficial for the enterprise. However software platforms such as BizTalk Server 2013, Mule ESB and IBM Websphere MQ have made it easier to create ESB systems. But the question of how web services can be efficiently integrated into such a model, and if it is beneficial for companies remains.

1.2 Problem Definition

Jupiter System Partner is currently working on a product idea called Information Logistics. The product will collect information via modules from several services in an enterprise system. The product will be based upon Microsoft Dynamics NAV[11], an enterprise resource planning solution (ERP), which can be customized via modules. Jupiter has already developed some prototypes of these modules. Amongst these, a module for online/offline functionality, and a possible underlying implementation for information handling. Figure 1.1 is an illustration of the system.

[9] `http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html` (collected 10.03.15.)

[10] `http://searchsoa.techtarget.com/definition/Remote-Method-Invocation` (collected 10.03.15.)

[11] `https://www.microsoft.com/en-us/dynamics/erp-nav-overview.aspx` (collected 07.05.15.)

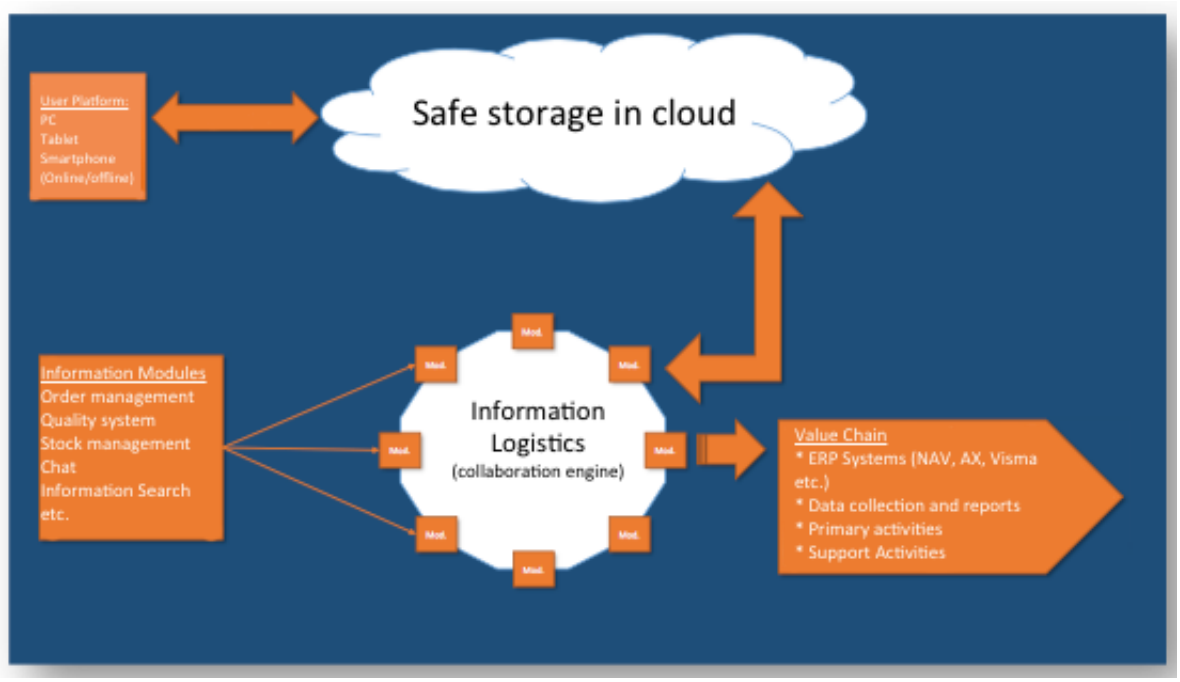

Fig. 1.1: Illustration of the envisioned system.

The user platform for Information Logistics will be PC/Mac, tablets and mobile phones. Information from modules will be stored in a secure cloud storage. The information modules will be; Order management, Quality management, Storage management, chat, information search, and more.

The project will focus on integration of several web services, where the end system is an integration platform where web services can be invoked from a singular access point. This master thesis will look into products, such as Mule ESB, BizTalk Server 2013, and IBM WebSphere MQ, which integrate several services to a singular access point. The products will be analyzed against each other and evaluated to determine the best approach for the system.

The project will focus on enterprises which constantly expands its web service portfolio. For these enterprises the benefit of having an integration platform which can be efficiently expanded is very important, since it will allow reuse of existing functionality. It is also useful since the web services need only be managed inside the integration platform, in cases where the individual web services are being moved or upgraded to new versions.

The project will integrate web services such as eSecretary, especially developed for Kraemer Maritime AS[12], which is part of the Information Logistics product idea. The resulting product will integrate web services running at customers, into a singular access point using BizTalk Server 2013. However, integrating web services into a singular access

[12] `http://www.kraemer.no/no/omoss/` (collected 26.05.15.)

point can be time consuming and tedious. This thesis will therefore look at how this process can be streamlined. This has lead to the following problem definition:

> **How can services be efficiently integrated in an Enterprise Service Bus (ESB)?**

The author wanted to look into this problem since creating a system which integrates web services can seem time consuming. The author wants to point out that services in this case points to web services. By looking at this problem, one would get a better overview of implementation and effort needed for creating a service bus. The problem would also show how an implemented service bus could be extended with several web services in a short timeframe.

This problem definition leads to the project of creating a system which allows integration of several web services into an ESB. The ESB would be implemented using BizTalk Server 2013, due to existing experience in Jupiter System Partner. The building platform was also chosen due to demand for developers with BizTalk experience in the authors home town.

The resulting system will integrate the eSecretary web service implemented by Jupiter System Partner, and a **ConversionRate** web service[13] which provides conversion rates between two currencies. The services will be integrated into a singular access point where the message specification and functionality will be provided. The resulting system will provide a basis for the Information Logistics product developed by Jupiter System Partner.

1.3 Contributions

The contributions made by this thesis are:

- An integration platform for web services.
- A system which publishes integrated web services to a singular access point.
- A presentation of different ESB building platforms.
- Evaluation of different approaches to integrating web services.

[13] `http://www.webservicex.net/ws/WSDetails.aspx?WSID=10` (collected 18.03.15.)

1.4 Limitations

In the work of this project, some areas are not taken into account.

In the development of the system there has been no focus on the security aspect of the system. The author recognizes this as an important part of the system, but due to time constraints have not been able to implement this into the system.

The system does not implement any user interface, but rather acts purely as a backend system for providing web service functionality. The end system will act as a basis for the Information Logistics product, and can be extended to provide a web interface for consumers.

1.5 Outline

The organization of the remainder of this thesis is as follows:

Chapter 2: Presents the background for starting this project, and work related to this thesis.

Chapter 3: Describes the ESB, SOA and BPE terms, and presents BizTalk Server 2013, IBM WebSphere MQ, and Mule ESB.

Chapter 4: Presents the design of Information Logistics Service Router as well as an experimental design in Mule ESB.

Chapter 5: Presents the implementation details of the Information Logistics Service Router system.

Chapter 6: Evaluates and discusses the Information Logistics Service Router system.

Chapter 7: Concludes this thesis and presents future work.

Chapter 2

Background and Related work

This chapter will discuss the earlier work and background to this thesis. It will elaborate on knowledge which has been achieved prior to and during the execution of this thesis. The chapter will end by a discussion of work which can be related to this thesis.

The work explained in this chapter acted as a basis for proceeding with the project, and decided what future work would have to be done to develop the Information Logistics product idea.

2.1 Pre-Project

This project is a continuation of a pre-project which was executed prior to this thesis. The pre-project looked at how data could be integrated into a common workspace or storage. The project looked into eSecretary[1] developed by Jupiter System Partner, which is a module for enterprise resource planning (ERP) solutions, such as *Microsoft Dynamics NAV*. The module makes spreading the secretary's information and overview over the organization easier. It is made to create a better workflow, remove obstacles and give a better overview.

The project looked into eSecretary and tried to find a suitable method to merge several companies storage from eSecretary into a common storage space. The project looked in depth at cloud architectures which resulted in a simple prototype where data could be stored in a single storage space. The data from companies would be identified with different company identifiers. The project concluded that the problem could be solved with either of the cloud architectures, and that choosing the correct cloud architecture would depend on the developing company's Service-Level Agreement (SLA)[2].

[1] `http://www.hoeghadmin.net/jupiterweb/e_sekretar.html` (collected 05.04.15.)

[2] `http://searchitchannel.techtarget.com/definition/service-level-agreement` (collected 04.05.15.)

This thesis is continuing the work from the pre-project. But rather than focusing on how the data can be integrated into a common storage space, it is looking closer into how this storage can be accessed from a common access point. The project is driven by the expected demand for applications and services utilizing integration solutions rather than point-to-point connections.

2.2 Existing knowledge/systems

The author of this thesis is and have now been employed at Jupiter System Partner for one year. The author approached the managers at Jupiter, in search for a problem which could be researched as a thesis subject. This resulted first in the pre-project and later in this thesis. The assignment was to help the business with a product idea; Information Logistics. The research on eSecretary from the pre-project generated knowledge of the eSecretary implementation, which was crucial for the execution of this thesis.

BizTalk Server 2013 was preferred as implementation platform due to existing knowledge at Jupiter System Partner. It was therefore decided to invest in a BizTalk introduction course[3] where the author could learn the basic functionality of BizTalk as a service bus platform. During this course the architecture of BizTalk was explained, e.g: how messages are passed through the service platform, and in which cases the software was fitting. Most user cases was pointed to an Enterprise Application Integration (EAI) platform where services from two or three different services are integrated together in a BizTalk application. The enterprise applications supply similar types of information to the BizTalk application, and then the application is usually used to transform or route the messages to supply the response.

2.3 Usage

The envisioned system was designed for Jupiter System Partner as part of a product called Information Logistics. The use of the system would be mainly to integrate web services that are used in this product. The integration of web services would help create web interfaces for accessing the eSecretary web service remotely. It would also help create a more robust product, where less effort would be needed to setting up the system at a product customer.

For this thesis, the author should investigate how the eSecretary web service and other web services could be integrated into an ESB. This would provide a good basis for the rest of the product implementation of Information Logistics, where the web service portfolio could be expanded as the product was implemented with more functionality. The

[3] `http://www.bouvet.no/Global/Kurs/Biztalk%20Immersion%202013%20produkt%20ark.pdf` (collected 26.05.15.)

system would also prove important in providing a Service-Oriented Architecture (SOA) for the enterprise.

2.4 Related work

Creating a system which integrates several services into a common end-point is not a new idea. In this section we will describe work which is related to this project. Each article will be presented and discussed related to how it compares to the envisioned system.

2.4.1 Building Scalable Enterprise Service bus

The ESB designed by Karim M. Mahmoud [3] illustrates how a unified architecture for building a scalable ESB can be achieved. The design includes the necessary components and message structure to implement an ESB where the process of integrating services requires very few, if any, changes to the original design. The system implements four components. The **Service Request Adapter** enables the system to use components whose interfaces don't quite match its requirements. The **Service Gateway** acts as a router for the request and the response messages. The **Atomic Services** handles all the needed message transformation, aggregation and composition. The **Service Provider Adapter** creates an abstraction layer for sending different request messages to the service providers and receiving response messages.

This ESB design would have fitted perfectly with the requirements for the desired service bus. But one of the system requirements was that the system was built on an existing ESB building platform. This implementation therefore does not fit the envisioned system since its implementation also develops a new message-based middleware. In this project we rather want to utilize existing ESB building platforms.

2.4.2 B2B Contracts using BizTalk

The system designed by Charles Herring and Zoran Milosevic [5] implements a Business-to-Business Contracts System. The system focuses on integrating data contract signing, since this may involve trading partners in several geographical locations. It also focuses on this since it can involve significant time and transaction costs associated with handling the contract signing process. The system can also provide a way to speed up the contract negotiation process.

The system design proposed by Charles Herring and Zoran Milosevic is implemented in BizTalk, which satisfies the requirement that the system is developed using an existing ESB building platform. But the system does unfortunately not implement an ESB,

but rather a Business Process Engine (BPE). Integrating the contract signing described involves a set of receivers of the data. The system is therefore more tightly coupled compared to an ESB design. This means that the system is implemented in such a way that the applications and services which are being routed in the BPE, is dependent on each other. In an ESB on the other hand, we only want to make the integrated web services accessible to other applications. The design however is one of the closer implementations compared to the envisioned design described in chapter 1, since its concept resembles what has been implemented in eSecretary and due to its use of BizTalk.

2.4.3 Versioned Web Services

The article written by David Frank, Linh Lam, Liana Fong, Ru Fang and Christopher Vignola [9] describes an implementation in IBM WebSphere ESB. The system implements a hosting service which can enable several versioned web services at the same time. The system routes incoming web service requests using the version metadata supplied in the web service request. The system design consists of three components (**Service Version Group Gateway**, **Version Management Controller** and **Version Group Configuration**) which collectively handles the routing to the correct web service.

The system designed by David Frank et al. implements a fully functional ESB using IBM WebSphere ESB. Compared to the envisioned system of this thesis, the Versioned Web Service Hosting service is implemented using IBM WebSphere ESB. For this thesis, we want to implement the integration platform on the BizTalk software. The proposed system design also focuses on managing Versioned Web Services, where we in this thesis want to focus on efficiently integrating web services.

2.5 Summary

This chapter has presented the background for starting this project as well as work related to the envisioned system. The chapter presented the process of going from a pre-project, which mainly focused on how data could be integrated into a common workspace or storage. It progressed to creating a system which could act as a basis for the product idea developed by Jupiter System Partner.

The next chapter will present different architectures and tools for developing such a system.

Chapter 3

Enterprise Service Bus Systems

This chapter will present different approaches to integrating web services, the terms connected to creating an ESB, and different ESB building platforms. This chapter will present BizTalk Server 2013, Mule ESB, and IBM WebSphere MQ. The chapter will end by presenting a new service integration technology, which although haven't been used for this thesis, can prove to be very useful when creating an ESB.

3.1 Service-Oriented Architecture (SOA)

The Service-Oriented Architecture (SOA) is an architecture pattern which allows applications and services to provide their business functionality and reusability to complex business processes that are implemented through a combination of several services.

SOA can be implemented using any service-based technology, such as SOAP or REST. The architecture can be utilized from enterprise applications and end-to-end business processes. It can be utilized even when the providers of those services are hosted applications running on different operating systems, written in different programming languages, or based on separate data models. The reason why many enterprises choose to utilize an ESB, is due to the recent focus on developing a SOA.

3.2 Enterprise Service Bus (ESB)

Utilizing an Enterprise Service Bus (ESB) has become a very attractive solution inside an enterprise. It is used to provide a reliable and efficient communication between services and applications. ESB works as a messaging architecture which handles protocol conversion, message format transformation, routing, orchestration, accept and delivery of messages from different services and applications. ESB also provides an approach for integrating applications, such as web services. ESB is a pluggable framework where

any new service can easily be integrated into the bus. Service providers and service consumers do not directly interact with each other, but rather communicate via the ESB which acts as a message broker between the applications.

An ESB is described by *MuleSoft* [24] as fundamentally an architecture for building a set of rules and principles. The architecture integrates numerous applications and services together over a bus-like infrastructure.

The idea of an ESB is to integrate different applications and services by putting a communication bus between them. This decouples systems from each other by allowing them to communicate without dependency on other systems in the bus.

The ESB architecture was introduced as developers wanted to move away from point-to-point integration, which can become tedious for system managers. An ESB can be used to leverage existing systems and expose them to new applications and services by using its communication and transformation capabilities.

3.3 ESB vs. BPE

There are two service integration models which enforces the SOA; BPE and ESB. ESB has already been described earlier in this chapter. The Business Process Engine (BPE)[1] however, is a software framework which provides business process interaction and communication between different data or processing sources to one or several applications and services.

The two Figures 3.2[2] and 3.1[3] illustrates the traditional architecture and implementation model of these two. As can be seen from the two figures the BPE model is a more tightly coupled implementation where the individual systems is dependent on the integrated client for its information. This architecture makes the individual applications brittle and hard to manage. The ESB figure on the other hand illustrates a model which is more decoupled from applications and web services, since integrated applications and services are not dependent on information from the ESB.

ESB and BPE both integrates applications and services through an integration client or system. The difference is that an ESB offers already existing applications/services, through an integration client, without having the applications/services knowing of this. BPE, however, creates a point-to-point connection between the application/service to the integration client. This creates a tightly coupled system, where the failure of one of the services, may cause total failure of the system.

[1] `http://www.techopedia.com/definition/26689/business-process-engine-bpe` (collected 20.04.2015.)

[2] `http://www.ibm.com/developerworks/library/ws-universalports-esb/Figure1.jpg` (collected 20.04.15)

[3] `http://www2.geog.uni-heidelberg.de/media/forschung/gis_ows_xnavi2.png` (collected 20.04.15.)

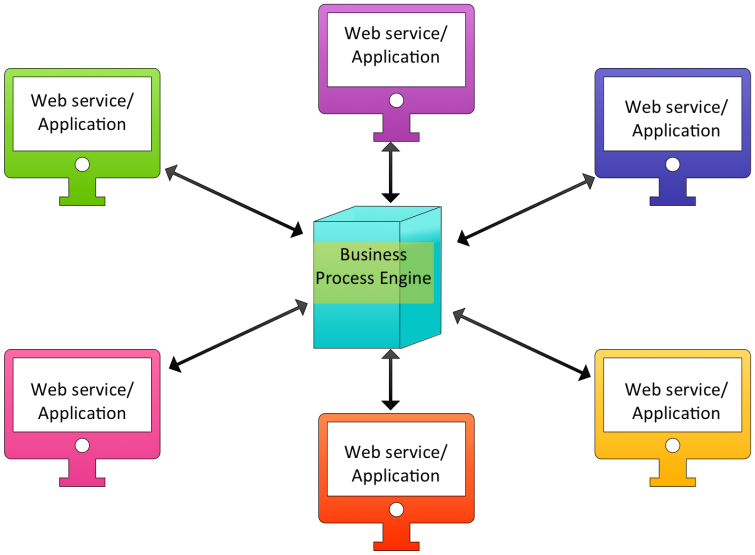

Fig. 3.1: Figure showing the traditional BPE architecture.

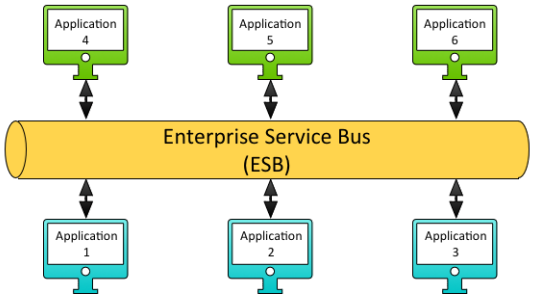

Fig. 3.2: Figure showing the traditional ESB architecture.

The ESB community and BPE community has long been in a conflict discussing what is the better technique for implementing SOA adaption programs. The paper by Thorsten Heller [10] illustrates the combination of traditional ESB and BPE components. According to Thorsten Heller, there are three major SOA usage patterns: **Service Oriented Human Interaction (SOHI)**, **Service Oriented Process Interaction (SOPI)**, and **Service Oriented System Integration (SOSI)**.

The SOHI pattern is usually applied in systems where many users are granted access to different business processes or services. The SOPI pattern is typically used when business processes are created across different companies, products or services to realize asynchronous and "long-running" processes typically implemented in a BPE. The SOSI pattern is usually applied for transactional and "short-running" processes, usually where focus is on enterprise application integration.

The paper by Thorsten Heller [10] proposes a consistent implementation model, illus-

trated in Figure 3.3, which defines a cooperation between the ESB and BPE models. The proposed model uses the ESB's service virtualization functionality to integrate the different layers within the SOA reference architecture. In this model, ESB is utilized for service virtualization and mediation, and BPE is utilized for long-running, horizontal and asynchronous business integration processes requiring human intervention. The latter is depicted in Figure 3.3 as "Integration flows" and "Business process flows", which is BPE's orchestration functionality. The ESB depicted in Figure 3.3 is used to glue together the different layers and utilize ESB's transformation and integration capabilities.

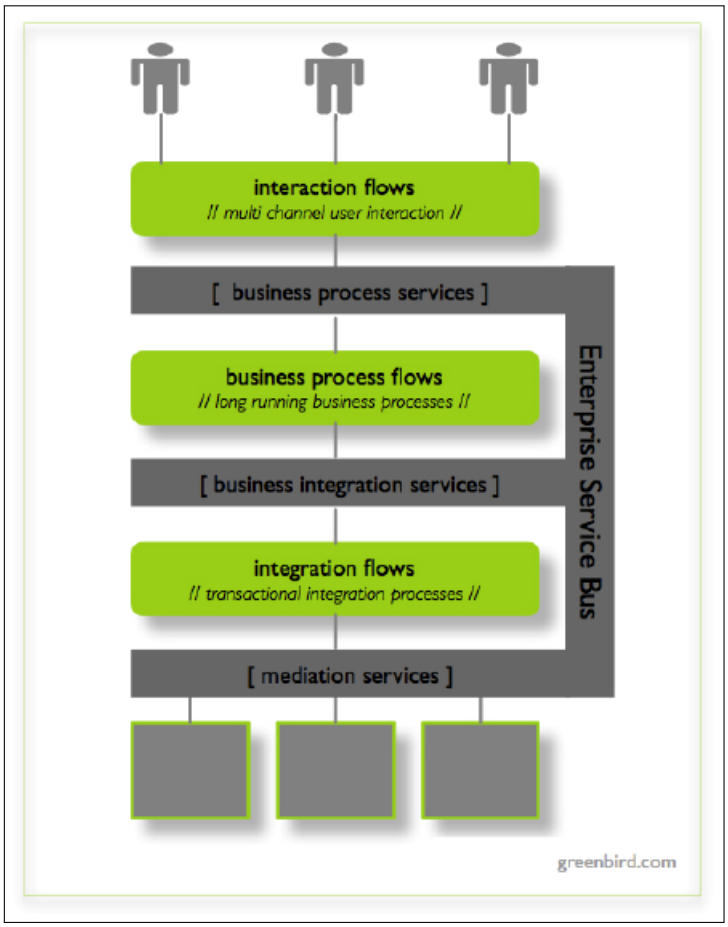

Fig. 3.3: Figure showing the combination of BPE and ESB architecture.

This combination will make the best of both integration architectures by using the ESB's integration and transformation capabilities to abstract the backend complexity, while the BPE is used to implement integration flows and business process flows by utilizing the orchestration functionality. This combination of functionality from the ESB architecture and the BPE is very useful for this project since the product will only be available to a certain group of users. It is also useful since the envisioned system should implement an ESB.

3.4 ESB building platforms

This section will present several ESB building platforms designed to integrate systems, and will present the functionality and architecture of the platforms. Two enterprise softwares will be presented, as well as an open source alternative. Finally the section will present an ESB platform, based on the Azure cloud platform developed by Microsoft.

3.4.1 Mule ESB

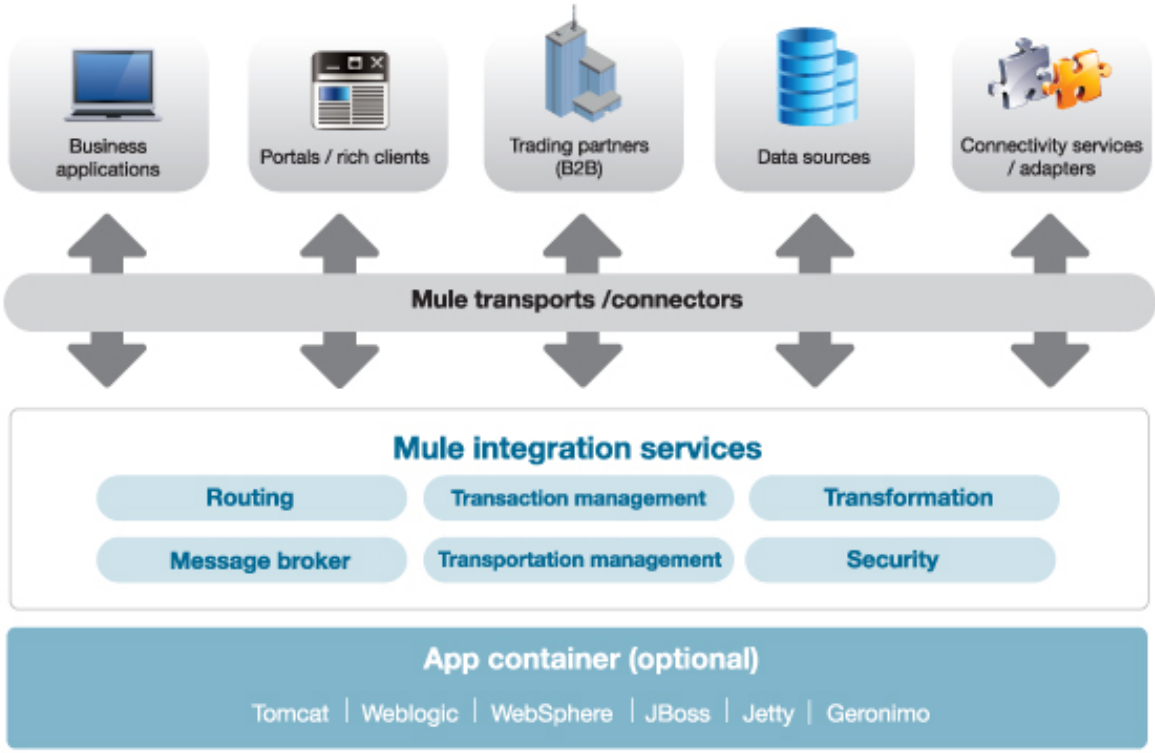

Fig. 3.4: Figure showing the Mule ESB architecture.

Mule is an Enterprise Service Bus platform designed for integrating legacy systems, applications and services without having to use the usual custom point-to-point integration. It offers a lightweight integration platform developed on the Java Runtime for creating service mediation, orchestration, message routing, data transformation and event handling.

Mule is a distributed Java-based enterprise service bus framework. It is designed to ease the task of integrating multiple enterprise components. It supports integration and reuse by enabling service components to be used in multiple different ways. It achieves this by using loose couplings between each service component through a variety of standard

messaging, transport, routing, connector, adaptor, aggregator, filter, security and transformer components, from the MULE framework. These will be acting as gateways and mediators. The Mule architecture is based on designing message flows, where end-point components, transformers and invokers are added. The messages are kept temporarily in memory, or streamed if a bigger file, while handling transforming, routing or invoking in the message flow design. This allows efficient routing of messages through the flow with minimal overhead. Figure 3.4[4] illustrates the Mule ESB architecture.

3.4.2 IBM WebSphere MQ

IBM WebSphere MQ is a messaging-based middleware (MOM)[5]. It packages information that is to be sent as a message and dispatches the message through a queuing system to a receiving application. The messaging system's logic enables it to be asynchronous. This means the application which is sending the information does not need to wait for a response, but rather sends the message through an API and continues doing some other work.

IBM WebSphere MQ provides a more robust, secure and reliable message passing for enterprises, by letting the messaging-based middleware ensure that messages are sent to the receiving application. The platform also provides transactional exchange of data, which ensures that messages are either collected correctly or not at all. This type of guarantee is especially interesting for bank applications, where data needs to be collected and updated on both sides of the transaction. If some part goes wrong in this transaction process, the message will be dropped and none of the sides are updated with a new state.

The platform also provides message persistence, ensuring that information is transferred even when there are system failures. Each message can, if required, be repeated or written to disk as a way to maintain the integrity of the information. This is helpful in cases where historical data is useful for applications and services utilizing the information. It can also be helpful for cases where transferred information will be revised.

Like other platforms presented in this thesis, IBM WebSphere MQ includes authentication to access implemented services in the platform. This could be used to let specific users access the implemented messaging systems. The platform also implements security measures of messages transferred between applications to avoid *man-in-the-middle* attacks[6], which are cases where the communication is being listened to by an unknown third party.

[4] `http://www.mulesoft.org/sites/all/themes/mulesoft_community/images/products/esb-lg.png` (collected 27.05.15.)

[5] `http://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html` (collected 04.05.15.)

[6] `https://www.owasp.org/index.php/Man-in-the-middle_attack` (collected 18.05.15.)

3.4.3 BizTalk Server 2013

The software chosen for this project was BizTalk Server 2013. The software was chosen due to existing knowledge at Jupiter System Partner, which has requested the system. BizTalk uses a *Message-box* architecture to perform message transformation, conversion, orchestration, assembling, and more. BizTalk is built on the Microsoft SQL Server database software to store information regarding schemas, orchestrations, messages, and more. It uses in total 9 databases to store runtime information checkpoints and more.

BizTalk is a powerful tool for integrating services and applications by using a message oriented architecture. The product allows implementing complex service and application integrations by utilizing a *Message-box* where data can be transformed and used to initialize orchestrations.

When fetching data to BizTalk, it can be parsed through a receive pipeline. In this pipeline data is disassembled into the wanted message format, either that be XML, flat-file, or other format. Likewise the data can be assembled into the subscribers wished format through a send pipeline. The receive and send formats are specified when creating the BizTalk application.

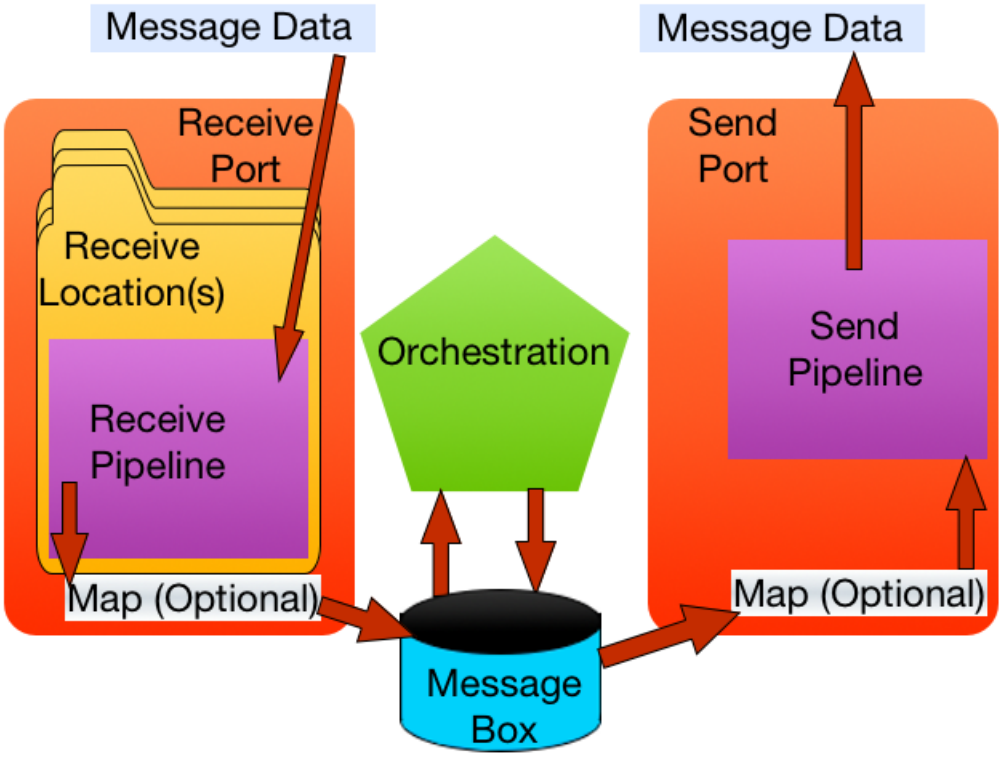

Fig. 3.5: Figure showing the BizTalk organization of elements.

The third crucial component of BizTalk is orchestrations. Orchestrations are computed arrangements or coordinations which use the data stored in the *Message-box* to compute

new data for subscribers or transforming data to the wanted output. Orchestrations can be used for decision making, transforming a message, changing individual properties of a message, sending and receiving messages through ports, and more. Orchestrations are an optional component which can be included in the BizTalk application.

By using BizTalk as a service bus allows companies to offer a specter of services through a single access point. It integrates services and filters the output depending on the data request and customer authorization. However, enabling this structure requires connecting the already existing services into the BizTalk application, which requires mapping the incoming data into some common known data form.

BizTalk is widely used for service integration. It's capabilities in message transformation, routing, orchestration and more makes it a very good choice for Enterprise Application Integration (EAI). The architecture is shown in Figure 3.5.

BizTalk is used in many settings as a service bus, and therefore Microsoft has created a separate Enterprise Service Bus Toolkit for BizTalk which is described in the next section.

BizTalk ESB Toolkit

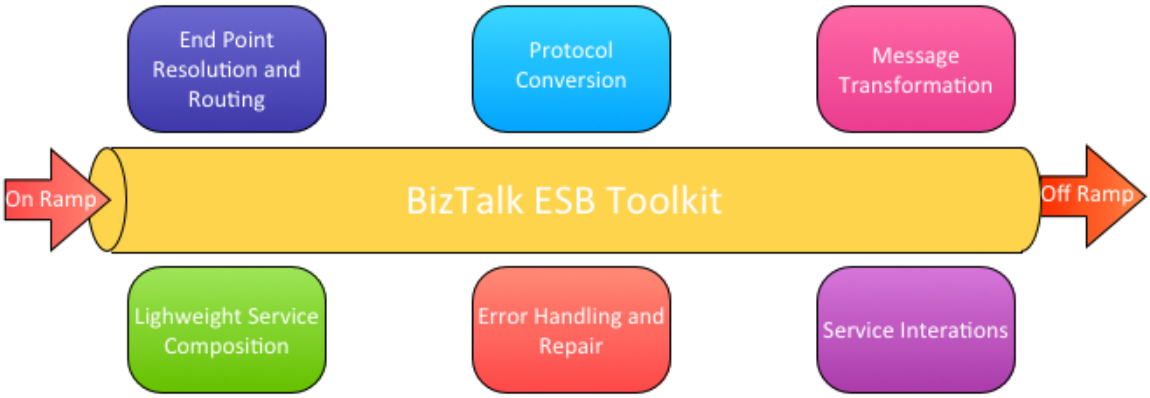

Fig. 3.6: Figure showing the architecture and components of BizTalk ESB Toolkit.

BizTalk has developed a toolkit[7,8,9] for integrating services in a singular service bus, which routes and transform messages to the end-service's format and destination. The toolkit implements several individual modules, which can be used to deliver the user request to the service provider. The resulting system can be used in a number of different ways. The system can for example be used for end-point lookup by using the end-point resolution and routing module. Transformation of message format to the end service

[7] `https://msdn.microsoft.com/en-us/library/ff699598.aspx` (collected 30.03.15.)
[8] `https://www.youtube.com/watch?v=tK6TQrKvPWw` (collected 24.03.15.)
[9] `https://www.youtube.com/watch?v=tNOdLrR43P0` (collected 27.03.15.)

format can be performed using the Message Transformation module. The architecture and components of the Toolkit is illustrated in Figure 3.6[10].

The BizTalk ESB Toolkit implements On-ramps, which are receive ports in a traditional BizTalk application but also includes pipeline components, special receiver components and interpreters. The ESB toolkit also includes Itinerary On-ramps, which act like a regular On-ramp but also includes an itinerary process. Itinerary processes are much like normal orchestrations and are, also in the BizTalk ESB toolkit, optional accessories to the service bus. Itinerary's are sets of processing instructions that travel with a single message. They are usually composed of one or more steps, referred to as itinerary services. These itinerarys are used by the ESB dispatcher pipeline component, and the *Message-box* to determine how to process the message. The itinerary is written to the message context when the message arrives at the On-ramp component.

Further the BizTalk ESB toolkit implements an Off-ramp component, which implements send ports for the delivery of messages. The Off-ramp can implement message delivery using formats and transports such as HTTP, JMS, WMQ, FTP, Flat File, XML, or other custom formats. The send ports are dynamic ports that are directly bound to the *Message-box*.

This toolkit is not used in the thesis since it only offers a higher level routing mechanism and not of the same level as in standard BizTalk. If this toolkit was to be used in the envisioned system it would only allow filtering of data, and not sending parts of the message to an integrated web service which is desired for the envisioned system.

3.4.4 Azure Microservices/BizTalk Services

A fairly new technology which Microsoft has developed is Azure Microservices. Azure Microservices, or BizTalk Services, is an integration platform part of the Azure Cloud technology implemented by Microsoft. The integration platform implements microservices which are hosted using Azure Websites. The Azure websites is part of Azure Cloud Service, which is an enterprise cloud that supports global scale and already runs millions of websites and API's. An overview of Azure Microservices is illustrated in Figure 3.7.

The services can be implemented in a wide range of programming languages, such as .NET, Java, PHP, Python and more. This allows flexibility for developers to create intelligent integration services in what language they feel most comfortable with.

Azure Microservices also allows processing calls between different Microservices by using Gateways. It utilizes this component to allow implementation of security, monitoring and governance in a centralized manner which helps developers create a better management connection between applications and services.

One important disadvantage of BizTalk server, is the complexity of deploying the application. According to Sam Vanhoutte [12] the applications implemented using Azure

[10] `https://social.msdn.microsoft.com/Forums/getfile/486521` (collected 18.05.15.)

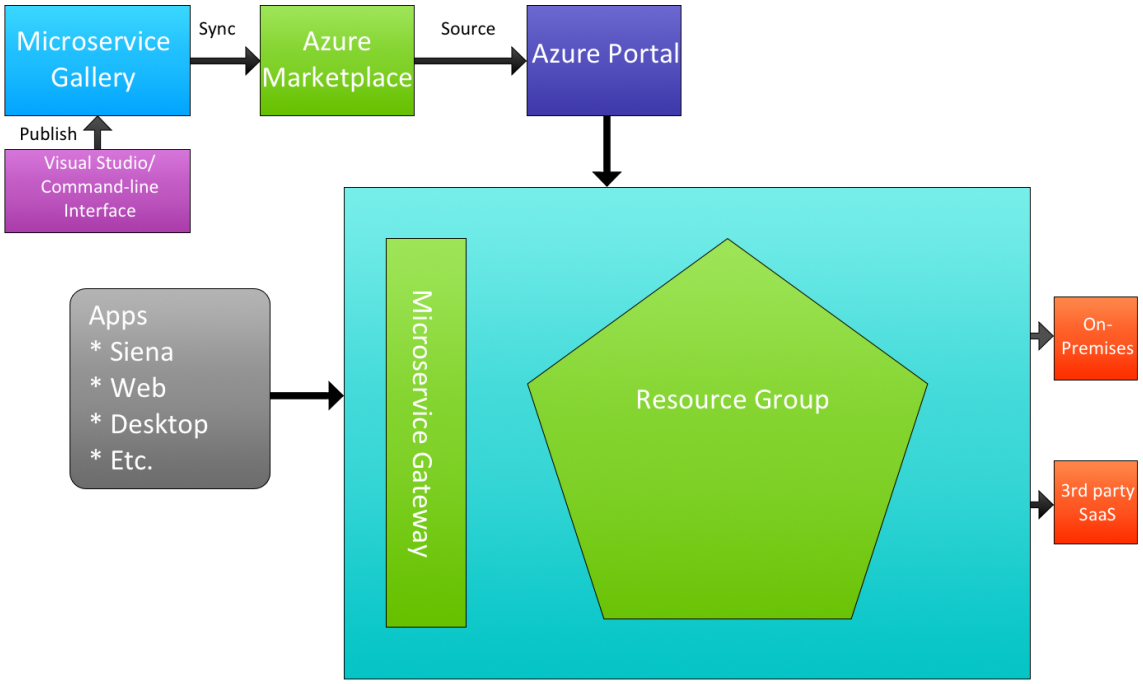

Fig. 3.7: Figure showing the overview of Azure Microservices.

BizTalk Microservices will be deployed using the Azure Resource Manager model, which simplifies the deployment procedure by dynamically managing number of workers, memory, etc. Scaling a system to the number of customers and users can be time consuming and trivial for the enterprise. By letting Azure Resource Manager dynamically scale and allocate necessary memory, processors and storage for the application/service might be very useful for especially smaller enterprises.

Implemented microservices can be accessed from the *Gallery*, which is the collection of all microservices created by the developer. The main point of Azure BizTalk Microservices is to utilize existing microservice apps to create new ones, whether they are using on-premises solutions (in the enterprise infrastructure) or cloud applications. The use of Azure BizTalk Microservices can improve the reusability of applications and services. Since the platforms' focus is to make services available, both from external developers and your own services, it will probably help developers which tends to create redundant functionality.

Azure BizTalk Microservices are also very flexible in terms of scalability. Azure allows a more fine-grained scaling of applications, or even auto-scaling, through the Azure platform.

In this thesis Azure BizTalk Microservices will not be utilized as it haven't been officially released. It is a promising technology which falls very well into the envisioned system. The platform is especially interesting since it primarily makes the ESB available from the

cloud, which makes it very portable and also makes the management of the service much easier. Microsoft Azure is well known for its well presented statistics of applications running in the cloud platform, which will make monitoring more straightforward. Also the use of *Gallery* can help create a better portfolio of the enterprises available services.

3.5 Summary

Software	**Open Source?**	**Pricing**
BizTalk Server 2013	NO	Enterprise Ed: $10,835 per core license, Standard Ed: $2,485 per core license, Branch Ed: $620 per Core license
Mule ESB	YES	Open-Source license
IBM WebSphere MQ	NO	Processor Value Unit License: $85, Telemetry Install License: $112, Advanced Message Security Component Processor Value Unit License: $29.75, Advanced Processor Value Unit License: $175, Advanced Developers Authorized User Single Install License: $2,740.

Table 3.1: Table showing the pricing comparison between the different ESB building platforms.

This chapter has presented different architectures and tools for building ESB systems and enforcing a Service-Oriented Architecture (SOA). The chapter presented three widely used ESB building tools for building an integration platform. The platforms' pricing are compared in Table 3.1[11,12]. As can be seen from the table, Mule ESB is an open-source software and therefore has no pricing. On the other hand BizTalk and IBM WebSphere are both enterprise software with different pricing. BizTalk offers different licenses which gives rights to deploy a certain amount of applications. The pricing is per machine. IBM WebSpheres' pricing offers different components of an ESB platform. So if an enterprise only wants the Processor Value Unit, they can buy that alone.

[11] `http://www-03.ibm.com/software/products/en/ibm-mq` (collected 12.05.15.)

[12] `http://download.microsoft.com/download/0/4/7/0479D5E5-021B-408E-8BCA-7D95AF76A9A6/BizTalk_Server_2013_R2_Licensing_Datasheet_and_FAQ.pdf` (collected 12.05.15.)

This chapter has also presented Azure Microservices, a platform built by Microsoft to move ESB implementations toward cloud platforms. This is a technology which looks very promising and might be very useful in future ESB implementations and also future work for Information Logistics Service Router.

The next chapter will present the design for Information Logistics Service Router, implemented in BizTalk Server 2013.

Chapter 4

Design

This chapter will present the design for Information Logistics Service Router. The chapter will describe what was planned for the system and how it was actually conducted in this thesis. The chapter will also describe how the system could have been designed in this thesis, and how this would affect the system. Inspirations to how the system was designed will also be discussed in this chapter.

4.1 Early Design Thoughts

One of the earlier ideas was to use the built-in SQL adapter[1] in BizTalk to collect information from the eSecretary document database. The SQL adapter in BizTalk provides functionality to collect information from a Microsoft SQL server[2]. By doing this, eSecretary could be optimized and reduced to a single service instead of routing to another service. The resulting eSecretary web service would then collect invoices, display them, and merge them with a collected signature. Figure 4.1 illustrates this design.

This design was desirable since it would act as the new eSecretary implementation. The main component of the Information Logistics product idea is the eSecretary implementation. Since the new product would consist of several modules which interact with the eSecretary system, it would be beneficial to have eSecretary part of the core system. Also the design would be desirable since it would allow the signature merging to be implemented into the new version of eSecretary.

Integrating the merging functionality into the eSecretary implementation in BizTalk would create a less complex system architecture where more of the functionality is com-

[1] `https://msdn.microsoft.com/en-us/library/aa560219%28v=bts.20%29.aspx` (collected 21.05.15.)

[2] `https://www.microsoft.com/en-us/server-cloud/products/sql-server-editions/overview.aspx` (collected 21.05.15.)

pressed into one service. This will be more compact compared to having the eSecretary and merging functionality separated from each other. The maintenance of the service would also be easier due to its compactness.

Another design was however chosen, since the complexity of implementing already existing functionality from eSecretary could not be accomplished in the time span of this thesis. Also the author wanted to place minimal functionality in the ESB implementation, since it would require less maintenance. Another important point was that the ESB implementation should not implement existing functionality, but reuse applications and services which implements this functionality. This is one of the main points of creating an Enterprise Service Bus (ESB). Creating this system would not follow the guide lines of a Service-Oriented Architecture (SOA), since redundant functionality would be implemented inside the enterprise, and services inside the service bus would depend on the eSecretary implementation. This would make the integrated web services brittle, which is what the SOA tries to avoid. When creating an ESB we need to make sure the integrated services are decoupled from the ESB implementation, since it allows them to act independent of the ESB.

Another reason to why the design was not chosen was due to the impact reimplementing the already existing eSecretary web service could cause. The articles [23, 25, 26, 27, 28] discusses the importance of software reuse. "Software reuse is a major way to boost software engineering productivity" states M.K. Zand and M.H. Samadzadeh in their article [23]. One of the important advantages mentioned in the article for software reuse is:

> *Improving system reliability by reusing components that have been (at least operationally) proven correct, and reducing the need for system testing.*

In this implementation, we have therefore chosen to follow the software reuse methodology and use the existing eSecretary web service since we know it is operational and don't need extensive testing.

4.2 Information Logistics Service Router Design

Figure 4.2 shows the design idea to utilize the already implemented web service for eSecretary, and create a service bus which would integrate the eSecretary web service and other web services into a singular access point. The service bus would be in charge of directing the individual requests to the correct web service. This would satisfy the principles of creating a SOA, where utilizing existing functionality is important.

The initial vision of this idea was to build several services running on different ports, since it would allow more easily management of each service or application. eSecretary would for example be running by itself and other services would be configured to run

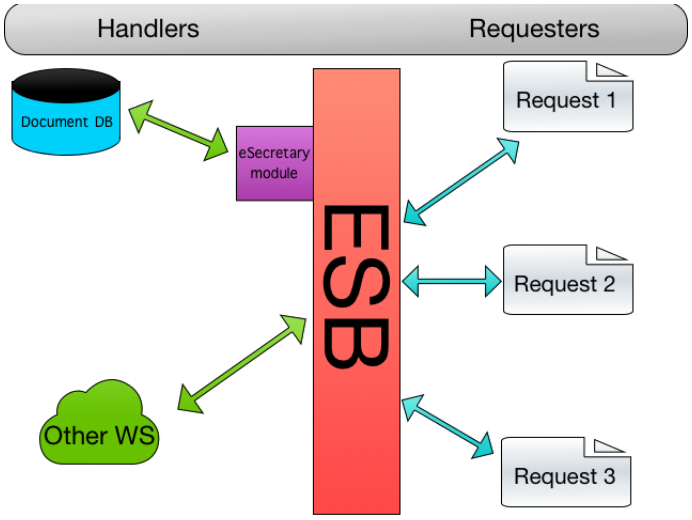

Fig. 4.1: Figure showing the design utilizing the SQL adapter of the Enterprise Service Bus.

on different ports. One of the main tasks of this project however, was to build an ESB system where integrated services could be accessed from a single end-point. By having services running on different ports, they would be accessible from the same system, but not the same port.

To solve this, the design was changed to an architecture where every service could be accessed from a single entry point, which would be important for the envisioned system. The system would otherwise function as having several web services and applications running on different access points. The single entry point would involve creating a routing mechanism which could pass the message to the correct web service depending on the message body. For example, would a eSecretary service request be passed via the routing mechanism to the eSecretary web service.

The routing mechanism would be the main component in the end system, since it would make sure that service requests where routed to the correct web service. However, it would also be the main overhead when creating the system, since it would involve routing and transformation of incoming service requests and invocations of the integrated web services. This would affect the overhead, since the routing would interpret and parse data in service requests.

The routing mechanism and single access point would be a key point when creating applications and web services which utilize the ESB implementation. Any application which would use information from one of the integrated web services, could connect to the ESB access point and run all the needed service requests to collect data. The routing mechanism would be implemented as an orchestration in BizTalk. An orchestration is an optional component which can be implemented into a BizTalk application. An

orchestration[3] can be used to design flows, interpret or generate data, call custom code, and visualize the process through an Orchestration Designer.

For applications using one of the integrated web services, this would mean easier management since the application would only need to handle one access point. In cases where the integrated web services changed access point, Information Logistics Service Router would need to be changed to point to the new web service end-points. However, the applications utilizing Information Logistics Service Router, would not need to make any changes to the initial access point unless the ESB implementation moved. This means less management for the application developers using Information Logistics Service Router.

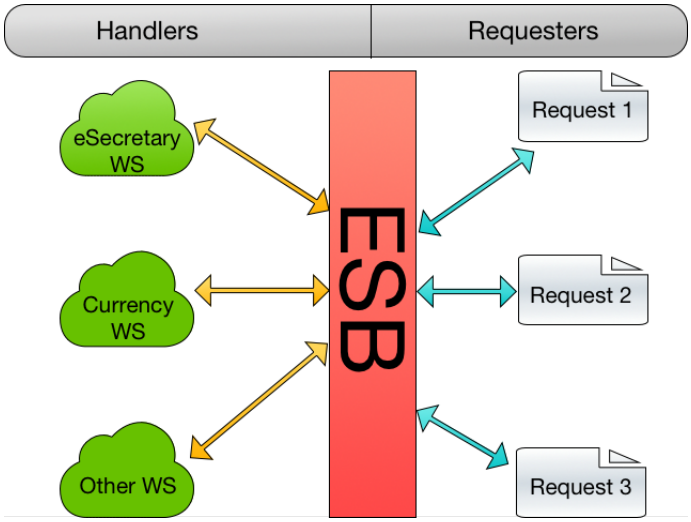

Fig. 4.2: Figure showing the initial design of the Enterprise Service Bus.

4.2.1 Schema design

The service schema is used by the BizTalk application to interpret the information being sent to the system. The service schema was created to help the routing mechanism interpret which integrated web service the individual service requests would be routed to. The schema was also created to have a common template for service requests being invoked on the ESB implementation. Without the service schema the system would not have been able to parse the web service requests that were sent through the service bus.

The service schema was designed comparable to what is proposed by Karim M. Mahmoud [3]. The schema includes a service identifier and a message body. The design was chosen so the service body and routing could be separated from each other. The service identifier is used to determine which web service will be invoked. The message body includes the operation structure of the web services, to enable easier mapping to

[3] `https://msdn.microsoft.com/en-us/library/aa578451.aspx` (collected 20.05.15.)

the web service schema. This would allow the user to place the function data in the message body, and identify the service or application to invoke by using the service identifier. For example the function *FindDocuments* from the eSecretary web service would be included as a record in the message body of the service schema. As would also other functions from other web services which are integrated into Information Logistics Service Router.

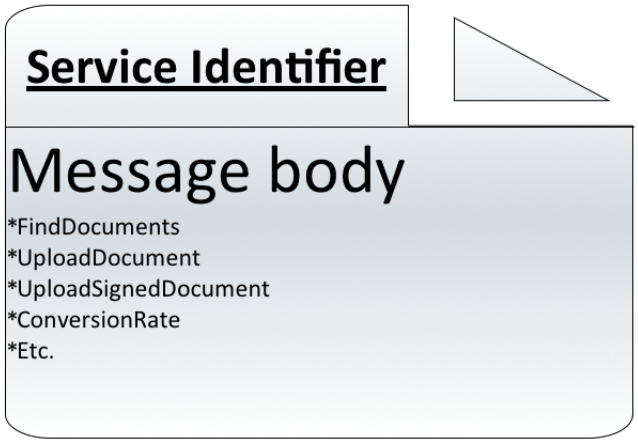

Fig. 4.3: Figure showing the schema design.

The initial idea was to take the data in the message body and pass it to the end-point web service. The only problem with such a design, is that the message cannot be mapped to the correct end-point service schema. The customer which is calling the service needs to also know the structure and parameters of the message, since the schema does not include any overview of which parameters the function takes. If for example function *func* takes in three parameters, it would not be visible for the consumer in the service bus schema.

To solve this, the schema was designed so consumers could keep track of which function they were invoking. The author decided that integrating the service schemas of the individual web services would therefore provide the best overview of functionality which is provided by Information Logistics Service Router. The service identifier field was also included in the schema design to easier route between the integrated web services. For example if a consumer would call the *FindDocuments* function from the eSecretary web service, they would have to choose the "FindDocuments" service identifier, and then add the necessary parameters for the function in the message body. The schema design is illustrated in Figure 4.3.

The web service schemas is generated using BizTalk's built-in *Consume Adapter Service*[4] component. This generates the web service schemas, web service port types and multi-part message types to use when invoking the respective web service. The purpose of

[4] `https://msdn.microsoft.com/en-us/library/bb798128.aspx` (collected 20.05.15.)

using the built-in component is to more easily create the needed schema files and end-point configuration for integrated web services.

The *Consume Adapter Service component* is a built-in functionality provided by BizTalk through the WCF LOB Adapter SDK[5]. The component creates orchestration and schema files from WCF Service metadata. The component can take in either metadata from a Metadata Exchange (MEX) point, or from a set of WSDL and schema files.

The designated service schema for the system causes extra maintenance. If any of the integrated web services are changed or extended with more functionality, then the service schema need to be changed to accompany the new or changed functionality of the web service. This means that system administrators need to keep track of changing functionality and update the service schema as the integrated web services are changed.

4.2.2 Orchestration Design

An orchestration was created to implement the routing functionality and invoke the integrated web services. The system would take in a service request including an identifier and a message body, which would include the message to the web service end-point.

The orchestration is the design's main component. It routes the message via a decide shape (illustrated in Figure 4.4) to the correct end-point web service. The decide shape examines the service identifier of the service request, and maps the message body to the correct end-point format. The mapping creates a service request using the web services' format. It then copies over the values from the Message body to the integrated web service request. The service request is then routed to the correct end-point web service through a collection of ports. BizTalk's built-in *Consume Adapter Service* component generates service ports with the web services' available functionality. It does this by utilizing the WSDL-file, which specifies the web services' message format and available functionality. The orchestration contains a collection of send and receive operations, one for each routing option, which will be invoked depending on the service identifier. The orchestration first sends the service request to the integrated web service, and then collects the response message. Finally the orchestration maps the service request back to the service response schema and delivers it to the ESB requester.

The service bus can be expanded by changing the service schema, the orchestration and adding necessary mapping files for transforming the service message to the correct end-point service schema. The enterprise can extend the initial integrated web services with more functionality by making these changes and create a greater portfolio of integrated services and applications.

For example, if the consumer wanted to integrate a new web service **A**, they would have to first use the built-in *Consume Adapter Service* component. Then the consumer would have to refactor the service schema to accompany the functionality provided by

[5] `https://msdn.microsoft.com/en-us/library/bb798128.aspx` (collected 20.05.15.)

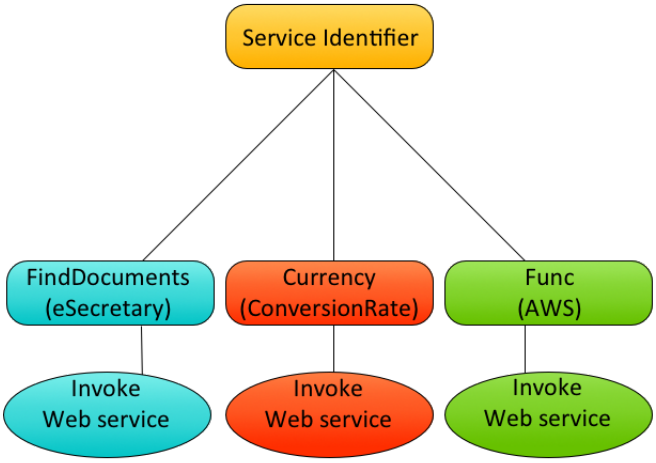

Fig. 4.4: Figure illustrating the decide shape.

the integrated web service, and also create mappings from the service schema to the web service format. Last the orchestration would need to be changed with additional routing options for the new web service functionality.

The orchestration inside BizTalk is an optional addition for an application. However, for this system it would not have been possible to route the message without an orchestration. The orchestration does add some overhead to the response time, due to the way BizTalk passes messages to an orchestration. The orchestration does timely polls from the BizTalk *Message-box*. The *Message-box* is BizTalks' database for service requests and messages. The interval between each poll from the *Message-box* creates a worst case overhead of 0.050 seconds where the orchestration is in a dehydrated state. This means the orchestration is doing no work. This is also the lowest poll interval for orchestrations in BizTalk.

4.3 Mule design

To evaluate the difference between integration softwares, an experimental system was designed in Mule ESB. The schema for the Mule ESB was designed differently compared to the BizTalk design which is discussed in the next section. Also the way routing, mapping and ports are implemented in Mule, is different from BizTalk. In Mule, rather than designing orchestrations, the flow for a message is designed. The way a message is received, transformed and routed was therefore designed differently from BizTalk. This would impact the evaluation, since the two systems were implemented differently. The prototype would integrate the **ConversionRate**[6] web service, which provides function-

[6] `http://www.webservicex.net/currencyconvertor.asmx?op=ConversionRate` (collected 15.05.15.)

ality to convert between different currencies.

4.3.1 Schema design

The ESB schema was created, equivalent of BizTalk, to interpret the information in the incoming service requests. The schema would provide the system with a template for information to be interpreted. Without a schema, the system would not be able to route, transform, or invoke the integrated web services. The schema was designed as a collection of operations or records. A separate record was added for each integrated web service operation. This meant that the **ConversionRate** web service operations together with the eSecretary web service operations, would be directly available in the service schema. For example if a consumer wanted to call function *func* from web service **AWS** it would be directly accessible by the consumer in the service schema. The consumer could therefore fill out the parameters in the same way as connecting directly to the integrated web service. The response would also be in format comparable to directly connecting to the integrated web service.

Compared to the BizTalk schema design, the Mule schema does not include a service identifier. It rather uses the name of the operation which is to be called, to route the message to the correct web service end-point. In comparison to the service schema designed in BizTalk, this meant that the schema would not include a separate service identifier field. The service schema would therefore be more well-presented compared to the BizTalk schema design, since the operations are directly accessible in the schema design rather than having them placed inside a message body. This meant that the routing would rely on the operation records' name for routing. If any of the operation record names were changed, the message flow would need to be changed to route the correct operation name. However, the structure of the service schema in Mule ESB is much easier to change compared to the BizTalk schema design. If a new operation should be added to the service schema, a new record can be added to the schema. The easier management of the service schema meant that the system could sooner respond to changing web service functionality.

4.3.2 Message Flow design

As mentioned, Mule is based on constructing **Message Flows** instead of orchestrations. The Mule ESB software requires a message flow to be implemented for an application. In the flow the ESB end-point is defined, together with the service type. For Mule ESB the service was designed as a Simple Object Access Protocol (SOAP) service, which is a language- and platform independent communication protocol to use for communicating between applications. Each web service operation is integrated into the message flow using the *Web Service Consumer* component. The component consumes a Web Service Description Language (WSDL) file, which defines the operation necessities, schema

structure, and message end-point. This means that having a well-formed WSDL-file would be important for future integrations. Each web service operation is defined in its own sub-flow, which is called from the main message flow. The sub-flow to call is decided in the choice component and depends on the called operation on the ESB implementation.

The message is parsed through the message flow in memory. This means that the extra overhead involved in orchestrations in BizTalk is non-existent in Mule ESB. Rather than collecting messages in a database, Mule parses them in memory or streams them while routing them through the implemented message flow.

For example if the function *func* from web service **AWS** was invoked, the operation name would be picked up in the message structure. The flow would then route the message depending on this operation name and call the appropriate sub-flow for function *func*, which would invoke the integrated web service.

Compared to the BizTalk design the separate sub-flow structure makes the system much more manageable and well-presented. This can be helpful when extending the ESB with more web service functionality, since minimal adjustments need to be done for the main message flow. Instead the invocations and transformations of data can be made in a separate sub-flow.

The two systems were designed differently since the Mule ESB implementation was used for evaluation purposes and due to differing functionality. In theory, the Information Logistics Service Router design allows better distinction by using a service identifier and a message body. It also allows a design where the message body can hold any data. But currently, Information Logistics Service Router uses a set of operation records in the message body.

4.4 eSecretary Web Service

The system was designed to utilize the eSecretary web service, which is designed and implemented by Jupiter System Partner. However the implementation as it is designed does not support filtering of several different companies. This means there is no functionality to distinct between several enterprises utilizing the web service. As mentioned in the pre-project [1], the eSecretary databases' fields include a **CompanyName** field. As far as the author knows, this field can be used to distinguish documents from several different companies from each other. The problem however, is that the web service implementation does not include filtering on this field when looking up documents. Due to this drawback, several eSecretary web services representing different document databases might need to be integrated into the same ESB.

With this design the ESB can include several eSecretary web services which all points to a different document database. This would then not enforce SOA, since there might be several eSecretary web services integrated into the same ESB system, creating redundant

functionality. It is therefore recommended that if this design is kept, then the eSecretary web service functionality should be changed to support several company identifiers and accommodate the increase in number of customers using the product.

With a design that supports several companies it would for example be possible to pass with a company identifier where documents from other companies would be filtered out. It would also be beneficial since each eSecretary system would not need to be integrated into the service bus, creating redundant functionality.

4.5 Summary

This chapter has presented the design for Information Logistics Service Router created in BizTalk Server 2013. The chapter has presented the schema design and orchestration design which defines how the service messages will be routed through the system.

The chapter has also presented the design for a Mule ESB prototype, used in comparison to the BizTalk implementation. The Mule design is designed differently compared to the main system due to different functionality in Mule ESB.

The chapter has also presented an early design idea, which was discarded due to time constraints and requirement for extensive reimplementation of existing functionality. The chapter also discussed missing functionality of the eSecretary web service which should be considered for future work on the system.

The next chapter will present the implementation for Information Logistics Service Router, as well as the implementation for the Mule ESB prototype.

Chapter 5

Implementation

This chapter will present the implementation of Information Logistics Service Router, describe into some depth the functionality of the implemented system, and to some degree show how it can be expanded with more services. The chapter will also describe an implementation in Mule ESB which has been implemented for evaluation purposes. Finally the chapter will describe the portability of the two implemented systems.

5.1 Early implementation

In an earlier version of the prototype the service was configured to utilize the BizTalk *SQL adapter*[1], which provides means to fetch data from the eSecretary document database. The version would reimplement a local version of the already existing eSecretary web service. The plan included collection of each separate document from the eSecretary document database and storing the data as XML documents together with a separate copy of the documents themselves. The XML document would include the company name, document type, what Enterprise Resource Planner (ERP) solution was used, the document itself, and more. The document was stored in the document database as a *Binary Large OBject* (BLOB)[2]. This means the data is stored as as a collection of binary data, in this case encoded in *base64*[3]. An example XML document collected from the SQL server is shown in Figure 5.1. This version was implemented since the optimal service bus structure would include the eSecretary implementation at its core. With this version the eSecretary web service could be integrated together with the document merger implementation, which is currently deployed at a different

[1] `https://msdn.microsoft.com/en-us/library/aa560219%28v=bts.20%29.aspx` (collected 21.05.15.)

[2] `http://searchsqlserver.techtarget.com/definition/BLOB` (collected 22.05.15.)

[3] `http://inchoo.net/magento/what-is-base64-encoding-and-how-can-we-benefit-from-it/` (collected 22.05.15.)

system. The document merger is the implementation which merges two documents together. In the eSecretary web service, this is used to merge a signature image together with invoice documents. The integration of both the eSecretary implementation and the merger would create a more compact system, which could be administered more easily. The cost of separate systems would also be less if these two were combined, since only one system would be needed.

The author however chose to use the design where the eSecretary web service would be routed in the Information Logistics Service Router. This would not require a separate storage space for documents. The early implementation would also require reimplementing the already existing functionality in the current eSecretary web service, including merging signature and the invoice document, and decoding documents to pdf format instead of **base64**. The design would also not enforce a Service-Oriented Architecture (SOA), where the focus is on reusing existing functionality. If this design was chosen, the system would implement redundant functionality. The author decided also that this implementation could not be achieved due to time constraints, and therefore chose the design explained in chapter 4.


```
▼<Document xmlns="http://EsekWS.com/Document">
   <Id>145</Id>
   <CompanyId>Customer</CompanyId>
   <DocumentType>5</DocumentType>
   <DocumentTypeId/>
   <ErpType>0</ErpType>
   <ErpId>70125</ErpId>
   <CustVendId>22953</CustVendId>
   <Document>BLOB</Document>
   <Mimetype>application/vnd.ms-outlook</Mimetype>
   <Filename>Epostelement.msg</Filename>
   <SizeInBytes>44032</SizeInBytes>
   <Inserted>2014-01-20T14:36:04.877</Inserted>
 </Document>
```


Fig. 5.1: Figure showing eSecretary XML document.

5.2 Information Logistics Service Router

The developed prototype implements a service bus in BizTalk Server 2013, which is an Enterprise Service Bus building software. The prototype utilizes a service schema, which can only contain a certain type of data. The service schema creates a template for the service bus to parse information in incoming service requests. This includes the request that is about to be routed to an integrated web service. The service schema includes a **ServiceId** field and a **Message** record. The **Message** record contains the message structure of each available operation in the service bus, which each can occur maximum one time for a service request. The **ServiceId** field is the string identifier which decides where the request is to be routed to. In the service schema structure there is defined one record per operation available for the integrated web services. This

means that if the **Message** field contains a record of one of these operations and it corresponds with the **ServiceId**, it will be picked up and parsed by the orchestration. The service schema structure and response is shown in Figure 5.2 and Figure 5.3.

```
<Service xmlns="http://ILServiceRouter.ServiceSchema">
  <ServiceId xmlns="">FindDocuments</ServiceId>
  <Message xmlns="">
    <FindDocuments>
      <erpType>0</erpType>
      <erpId>1234</erpId>
    </FindDocuments>
  </Message>
</Service>
```

Fig. 5.2: Figure illustrating an example service message.

```
<ns0:Service xmlns:ns0="http://ILServiceRouter.ServiceSchemaResponse">
  <Message>
    <FindDocumentsResponse>
      <FindDocumentResult>[{"CompanyId":"KRAEMER","CustVendId":"22890",
        "CustVendName":null,"Data":null,"DocumentType":"signert folgeseddel","DocumentTypeId":"id999",
        "ErpId":"1234","Filename":"signert_testkontrakt3.pdf","Id":"2",
        "InsertedDate":"3\/19\/2011 7:05:47 PM","Mimetype":"application\/pdf","Size":"0"},
        {"CompanyId":"KRAEMER","CustVendId":"22890","CustVendName":null,"Data":null,
        "DocumentType":"signert folgeseddel","DocumentTypeId":"id999","ErpId":"1234",
        "Filename":"signert_testkontrakt3.pdf","Id":"1","InsertedDate":"9\/21\/2011 10:34:15 AM",
        "Mimetype":"application\/pdf","Size":"0"}]</FindDocumentResult>
    </FindDocumentsResponse>
  </Message>
</ns0:Service>
```

Fig. 5.3: Figure illustrating an example service response.

The service implements a *Windows Communication Foundation* (WCF)[4] service access point, where the ESB implementation is supplied. WCF is a tool often used to deploy services and applications which maintains the SOA. The WCF service places the service messages into the BizTalk *Message-box*, which stores all messages passing through the BizTalk Server system. The service request is then collected by the implemented **ILServiceRouter** orchestration which then routes the message depending on the **ServiceId**. An example service request is depicted in Figure 5.4. The BizTalk implementation consists of three projects, one for each integrated web service and one for **ILServiceRouter**. The projects for each integrated web service consists of the files generated from the built-in *Consume Adapter Service*[5] component. The *Consume Adapter Service* component is a built-in functionality provided by BizTalk. The component creates

[4] `https://msdn.microsoft.com/en-us/library/ms731082%28v=vs.110%29.aspx` (collected 20.05.15.)

[5] `https://msdn.microsoft.com/en-us/library/bb798128.aspx` (collected 20.05.15.)

orchestration and schema files from WCF service metadata. The component takes in metadata from either a Metadata Exchange (MEX) point, or from a set of WSDL and schema files.

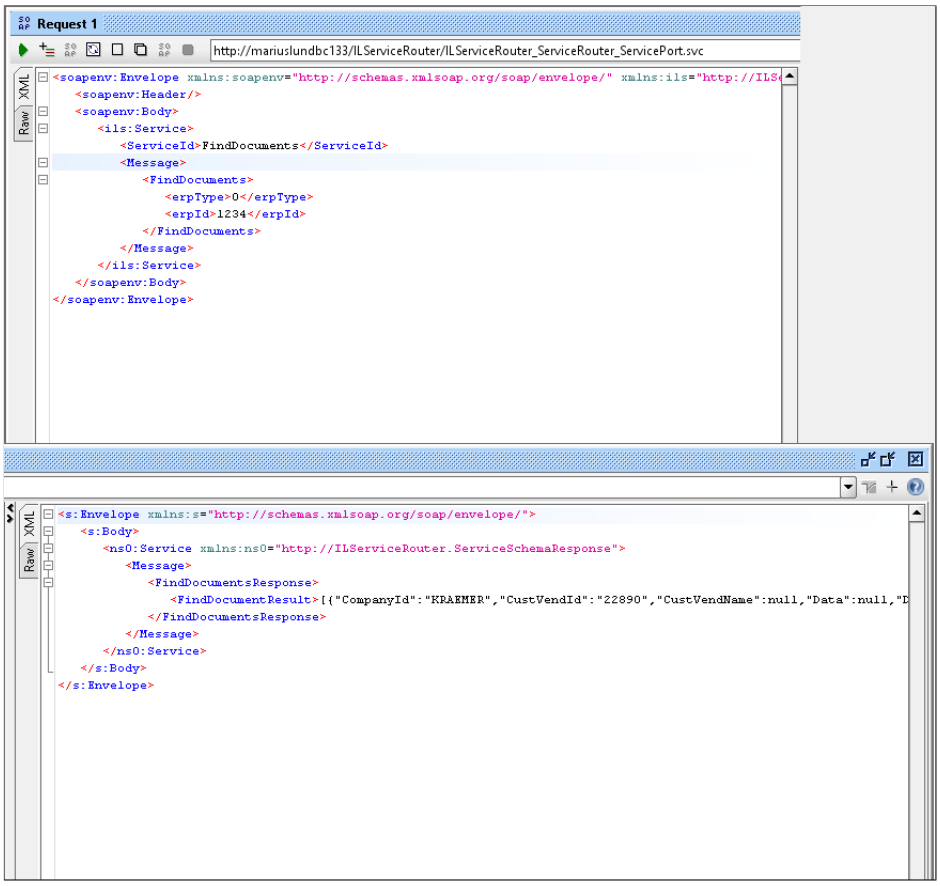

Fig. 5.4: Screenshot of a service request performed in SOAPUI.

Service messages are placed in the *Message-box* database in BizTalk. The orchestration does polls for messages which satisfies its schema requirements every 0.050 seconds. When it finds a message it collects and parses it through the orchestration. This routes the message to the appropriate end-point web service. After the message has been parsed through the orchestration, it is placed back into the *Message-box* database.

The WCF service is created using the built-in *BizTalk WCF Service Publishing Wizard*[6]. The wizard allows developers to publish either the implemented schema or orchestration as a WCF service. The wizard creates an internet application in Internet Information Services (IIS)[7] and creates a binding which is placed in the application and can be configured in the BizTalk Administration Console. For Information Logistics Service Router, we have published the orchestration as a WCF service using the WCF-WSHttp

[6] `https://msdn.microsoft.com/en-us/library/bb226547.aspx` (collected 20.05.15.)
[7] `https://www.iis.net/overview` (collected 20.05.15.)

adapter[8]. This is an adapter to perform network communication with services and clients that can understand next-generation web service standards. The wizard uses the application *.dll*[9]-file to examine the port type, namespace, and application functionality. The *.dll*-file is a file which contains instructions that programs can call upon to do certain things.

In the orchestration the service message is mapped to the service end-point schema structure. It then sends the resulting message from the mapping to the end-point service, collects the response and maps it back to the implemented service response schema. The resulting service response is then sent back to the client requester. A figure illustrating the **ILServiceRouter** orchestration is depicted in Figure 5.5.

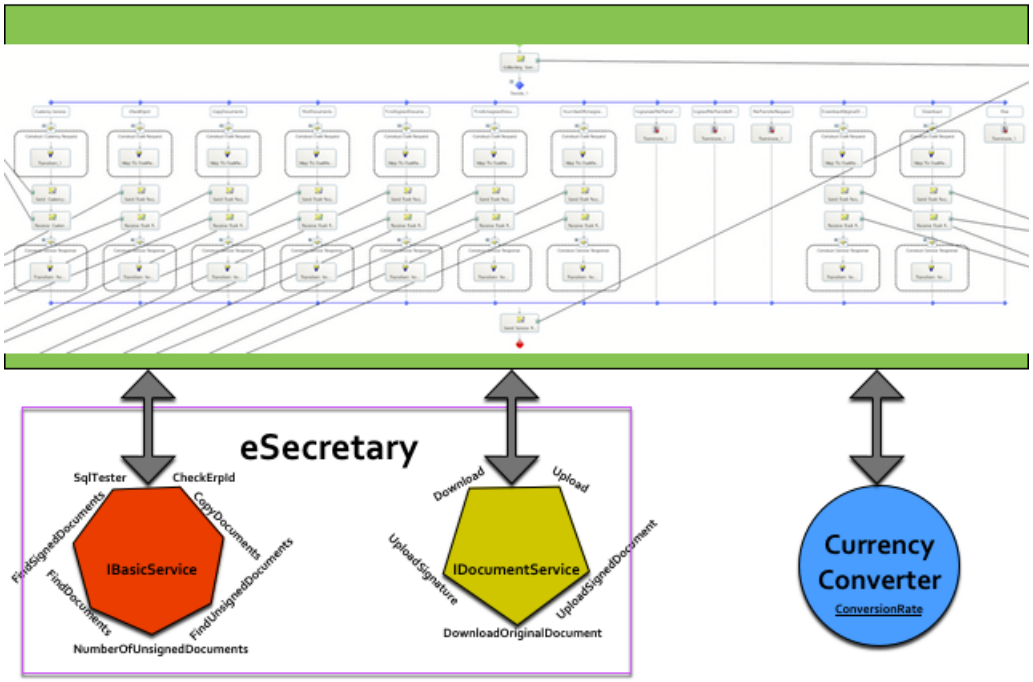

Fig. 5.5: Figure showing the current design of the Service bus with Orchestration.

If the ESB implemented in BizTalk needs to be extended, the orchestration and schema will need to be changed to incorporate the web service operations. The integrated web service need also be generated through the BizTalk *Consume Adapter Service* component to allow use of the web service ports and mapping to service messages.

Also to deploy the new solution, the existing application will need to be stopped and deleted, or configured as a new solution, using the BizTalk Administration Console. This would mean that if any changes was made to the BizTalk application it would need to

[8] `https://msdn.microsoft.com/en-us/library/bb245971.aspx` (collected 20.05.15.)
[9] `http://pcsupport.about.com/od/termsd/g/dll_file.htm` (collected 20.05.15.)

be deployed as a new application, or the current would have to be stopped and deleted before the new was deployed. This means the enterprise hosting the application either needs a license which allows several running applications on one system, or compensating for a calculated downtime of the system.

5.3 Mule ESB prototype

A second prototype was implemented in Mule ESB[10], which is another ESB building software. The prototype in Mule ESB was implemented to evaluate how creating an ESB in different softwares compares in terms of performance, efficiency, and usability.

A message flow was implemented to specify how a service request is to be parsed in the system. Mule uses message flows to plug together a series of message processors. The typical message flow has a message source, which accepts messages from an external source and triggers the execution of the flow. It also typically includes a series of message processors which transforms, filters, and enriches messages. The message flow for the implementation included an HTTP endpoint component as message source, which made the flow available as an HTTP service. The HTTP endpoint adapter was used in collaboration with the Simple Object Access Protocol (SOAP) component, where the service was configured with an appropriate WSDL-file representing the different operations that could be invoked on the service bus. The SOAP component would make the service available for cross-computer communication.

The prototype implements a service schema to provide a template for how messages are passed through the message flow. It contains a set of operations which are available by the integrated web services. The service schema was implemented using Mule's built-in schema designer which provides a visual design of the schema. Screenshots of the schema designer in Mule ESB is illustrated in Figure 5.6.

The message flow was then configured to collect the operation identifier. The operation identifier represented the name of the integrated web service operation that would be invoked. The integrated web service operations were represented in the service schema as a series of schema records. The message was routed using a **Choice** component, which utilizes the operation identifier to invoke the appropriate sub-flow[11]. A sub-flow is a flow which synchronously processes messages. The appropriate sub-flow would be in charge of sending the appropriate request to an integrated web service. The main message flow would pause while the sub-flow was executing, until it had collected a response from the integrated web service.

The integration of the web services was done using a **Web Service Consumer**[12] component, which utilizes a connector configuration and a specified operation to be

[10] `http://www.mulesoft.org/what-mule-esb` (collected 13.04.15.)

[11] `http://www.mulesoft.org/documentation/display/current/Flows+and+Subflows` (collected 21.05.15.)

[12] `http://www.mulesoft.org/documentation/display/current/Web+Service+Consumer` (col-

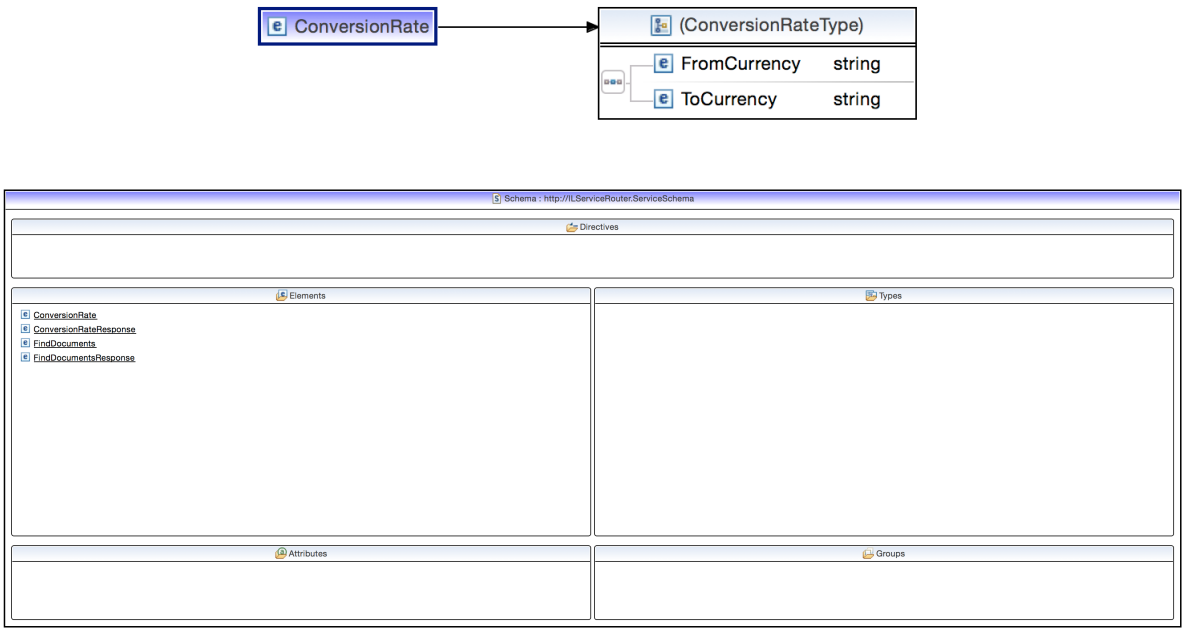

Fig. 5.6: Screenshots showing the schema editor in Mule ESB.

called. The connector configuration specifies the settings using a WSDL-file, or location, and configures the service, port and address using that same WSDL. The web service consumer component also handles the sending and receiving of the message by utilizing the WSDL to acquire information about the expected format and structure of the request and response. The response from the web service is then sent back to the requester concluding the message flow.

The **Web Service Consumer** component is used in conjunction with the **DataMapper**[13] component, which transforms the data structure and format to produce the web service's expected input. The **DataMapper** component makes use of built-in functionality in Mule ESB to acquire information about expected input for the web service through the **Web Service Consumer** component.

The implementation does not integrate the eSecretary web service due to its lack of a well-formed WSDL-file explaining the services' operations and request/response message. Mule requires this to invoke the external web service. The web service was therefore not integrated into the ESB implementation, since it would require implementing potentially a new web service.

The lack of the eSecretary web service, makes the prototype unfitting for the envisioned system. But as mentioned earlier this prototype will be used for evaluation purposes to see how BizTalk and Mule perform against each other. Since the eSecretary web

lected 21.05.15.)

[13] `http://www.mulesoft.org/documentation/display/33X/DataMapper+Transformer+Reference` (collected 21.05.15.)

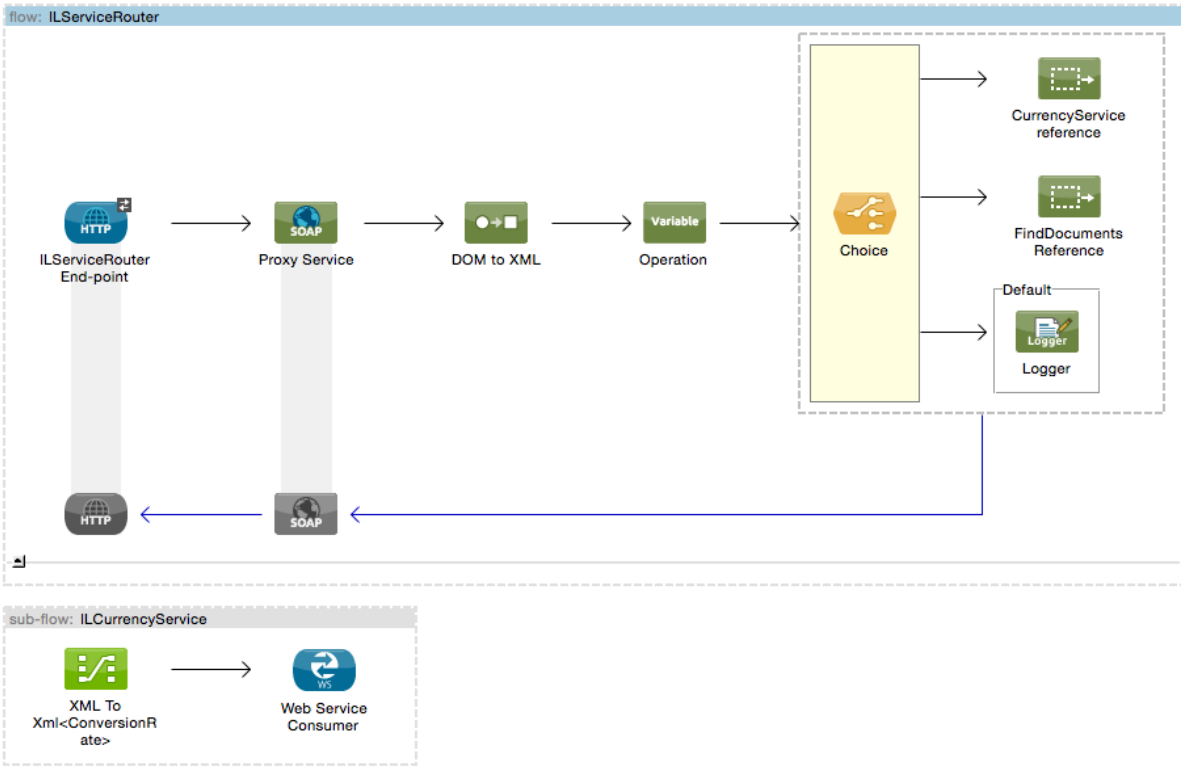

Fig. 5.7: Figure showing the service flow of the Mule ESB implementation

service is not integrated into the prototype, it does mean that the **ConversionRate** web service will be the decisive factor for the evaluation results. The results of this evaluation will be presented in the next chapter.

The ESB implemented in Mule can be extended by making a separate sub-flow for the added web service. The choice shape will need to be changed to incorporate the new web service. The schema will also need to be changed if the web service utilizes a completely different operation compared to the **ConversionRate** web service. It can in other words be quite trivial to extend the service bus since many components need to be changed.

For example if web service **AWS** was to be integrated into the ESB implementation, one would need to create a sub-flow for each operation available in web service **AWS**. If for example the web service had an operation *func*, it would have its own sub-flow. The sub-flow consist of a *DataMapper* component, which maps the fields of the service schema to the web service input. Also the sub-flow would consist of a *Web Service Consumer* component, which uses a WSDL-file and creates web service ports, message types, etc. The clue here is that the *Web Service Consumer* component will take care of the service invocation and produce a result back to the caller. For the integrated web service **AWS** the ESB schema would need to be changed to accompany the operations available in the web service, such as operation *func*. Finally, a separate choice branch

would need to be created for the different operations, such as *func*.

5.4 Portability

The two implementations presented in this paper, requires little adaptation to work on different computing systems. Information Logistics Service Router requires a Windows operating system and an installed version of BizTalk Server 2013 R2. The implementation is configured to use the Microsoft SQL Server[14] installed on the testing environment, which is a database software developed by Microsoft. Thus if the implementation was to be used on a different system the connected SQL server would have to be changed. The integrated eSecretary web service is also using a local version running on the test environment. To be able to port the implementation, this integrated service would have needed to point to the new service instead of the local test version.

For the Mule implementation, no changes would have been needed since the implementation does not make use of any dedicated database and since the server does not integrate a local version of the eSecretary web service. The implementation can be run on most operating systems available. In terms of portability the Mule implementation, though it does not integrate the same amount of services, is easier to manage since it lacks the dedicated database.

For example if the Mule implementation was to be moved to a new server, no changes would have to be done to deploy the system. In comparison, Information Logistics Service Router would need a new SQL server, or connect to one via the network, to function.

As mentioned Information Logistics Service Router requires both a version of BizTalk installed on their machine, and access to a SQL server where the necessary databases for BizTalk can be created. This might be undesirable for enterprises which require minimal impact on their infrastructure. BizTalk itself requires a 1 GHz or higher single processor for example. This means that a separate server will need to be set up for the system. Also if the enterprise don't already have access to an SQL server, this will also need to be set up. This means that the system might require, at most, 2 separate servers for software. Also the software is not open source, meaning the enterprise will need to invest in software to run the system. The system can, in other words, cost the enterprise very much to set up.

[14] `https://www.microsoft.com/en-us/server-cloud/products/sql-server-editions/overview.aspx` (collected 21.05.15.)

5.5 Summary

This chapter has presented the implementation of Information Logistics Service Router, which is implemented using BizTalk Server 2013. The chapter has explained how messages are routed using an orchestration, and presented the service request and response message structure. The chapter has explained the Mule ESB prototype implementation and how it differs from Information Logistics Service Router. Finally the chapter has discussed the portability of the two systems and compared them against each other.

Although an important part of any ESB system, this thesis have not implemented any security measures other than what is provided by the software and integrated web services. The author however, considers this an important part to future work on the implementation.

The next chapter will evaluate the two implementations and see how they compare against each other in performance and usability. Last it will evaluate how the system, and platform, solves the problem definition.

Chapter 6

Evaluation

This chapter will present measurements and evaluation of the Information Logistics Service Router prototype. The system will be evaluated against the prototype implemented in Mule ESB to see how the two implementations differ. The chapter will also discuss possible future work and improvements which could be added to the prototype, and conclude to see how this thesis solves the problem definition.

6.1 Testing Environment

The tests was executed on two different computers. The BizTalk implementation was tested on a quad core Intel i7-3770k CPU with 3.4 Ghz, 8 GB RAM, Windows 8.1 64-bit operating system, and a wired network connection of 25 Mbit/s download and 10 Mbit/s upload rate.

The Mule ESB implementation was tested on a quad core Intel Core i7 2.5 Ghz CPU, 16 GB RAM and OSX Yosemite version 10.10.3 operating system with a wireless network connection of 17 Mbit/s download and 10 Mbit/s upload rate.

6.2 Results

Information Logistics Service Router was tested executing a **C#** script running up to 1000 sequential requests through the service bus and directly to the web services. As can be noticed in Table 6.1 we see that the service bus adds a great deal of performance degradation. This is due to how BizTalk handles messages that need to be routed through an orchestration. BizTalk is a message-based system where messages are stored temporarily in a *Message-box*. When a request is delivered to BizTalk, it stores this as a message inside the *Message-box* database. Later, depending on the polling interval, BizTalk collects this message to be routed through the **ILServiceRouter** orchestration.

A great performance degradation was experienced due to this extra overhead. It could be especially noticed from the eSecretary web service, which handled 100 sequential requests in just over 1 and a half second. The BizTalk implementation handled the same amount in over 1 and a half minute. This means that there was roughly a 60x performance degradation when utilizing BizTalk orchestrations.

The author made some optimizations after a few searches on optimizations for BizTalk orchestrations and looking closer into the BizTalk Administration console. First off, the physical and virtual memory usage was boosted to utilize 90-100 % of the available memory. This enables the BizTalk application to utilize more of the available memory for messages. Next up the polling intervals for messages in the *Message-box* and orchestrations was set to the minimal amount (50 ms) which meant that messages would more quickly get picked up by BizTalk. Finally the number of subscriptions was set to pause at 100 and resume at 50. The latter specifies the number where the message-box will stop and resume sending messages to the orchestration. In this case, it will pause when the orchestration has 100 outstanding messages, and resume when it has reached 50 outstanding messages.

These optimizations was able to reduce the response time from 103 to 50 seconds, approximately halving the response time of the service bus. The response time for **ILCurrencyService** is pretty close to the original web service, as can be seen from Figure 6.1, but response time for the **ILEsekService** is still much slower than the original web service.

One thing to notice here though, is that the eSecretary web service is running locally while the **ConversionRate** web service is running over network. The response time when interacting with services over network might be congested with disruption and other jitter that reduces the response time. This might for example be loss of message parts in transmission to the web service, resulting in the message having to be resent. From Figure 6.1 this can be noticed since the eSecretary web service handles 1000 requests in roughly **4.8** seconds, while the **ConversionRate** web service handles the same amount in **709.7** seconds. This might be the reason why the **ILCurrencyService** response times are much closer to the original service. Thus we might see a closer comparison of a eSecretary web service running over network.

Requests	EsekWS	ILEsekService	ConversionRateWS	ILCurrencyService
1	00:00:00.67	00:00:03.07	00:00:01.63	00:00:03.62
10	00:00:00.77	00:00:11.65	00:00:07.04	00:00:19.37
100	00:00:01.66	00:01:43.54	00:00:59.78	00:02:15.62
1000	00:00:10.07	00:17:39.63	00:10:28.52	00:26:29.00

Table 6.1: Table showing the response times (in Hour:Minutes:Seconds) for the Information Logistics Service Router implementation.

The reasoning for the extra overhead experienced from the BizTalk implementation is

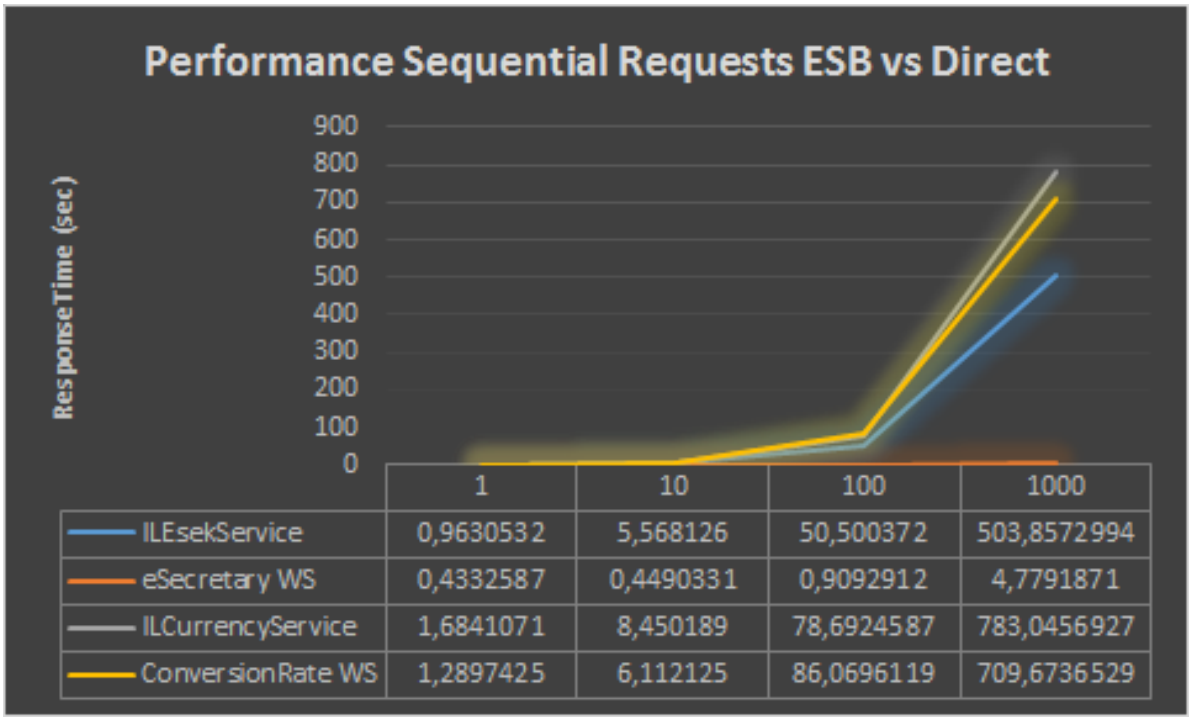

Fig. 6.1: Figure showing results after optimizations.

due to the message flow in the BizTalk architecture. When a message is received in BizTalk it passes through a receive pipeline and is stored in the BizTalk **Message-Box** component. This component is the heart of the BizTalk architecture where messages are stored and collected by send ports and orchestrations. This is also the reason why BizTalk's message flow isn't necessarily efficient. When a message stored in the *Message-box* satisfies the filter of an orchestration, it collects this message and parses the message through the orchestration and places it back in the *Message-box*. This is mainly why we saw a great improvement when reducing the polling rate for orchestrations in BizTalk. The extra time added when saving temporarily versions of the message in the *Message-box* is what is observed from the results in Figure 6.1 and Table 6.1.

6.2.1 Performance BizTalk vs. Mule ESB

The evaluation tests for Mule was ran using SOAP UI[1] by connecting to the **ConversionRate** web service and Mule ESB end-points. Then several independent requests was invoked on the respective services which produced the results shown in Figure 6.2 and Table 6.2. As can be seen from the figure and table there is an average 2x longer response time compared to invoking the web service directly.

When comparing the Mule response times with BizTalk, we notice that the same de-

[1] `http://www.soapui.org/About-SoapUI/soapui-faq.html` (collected 27.04.15.)

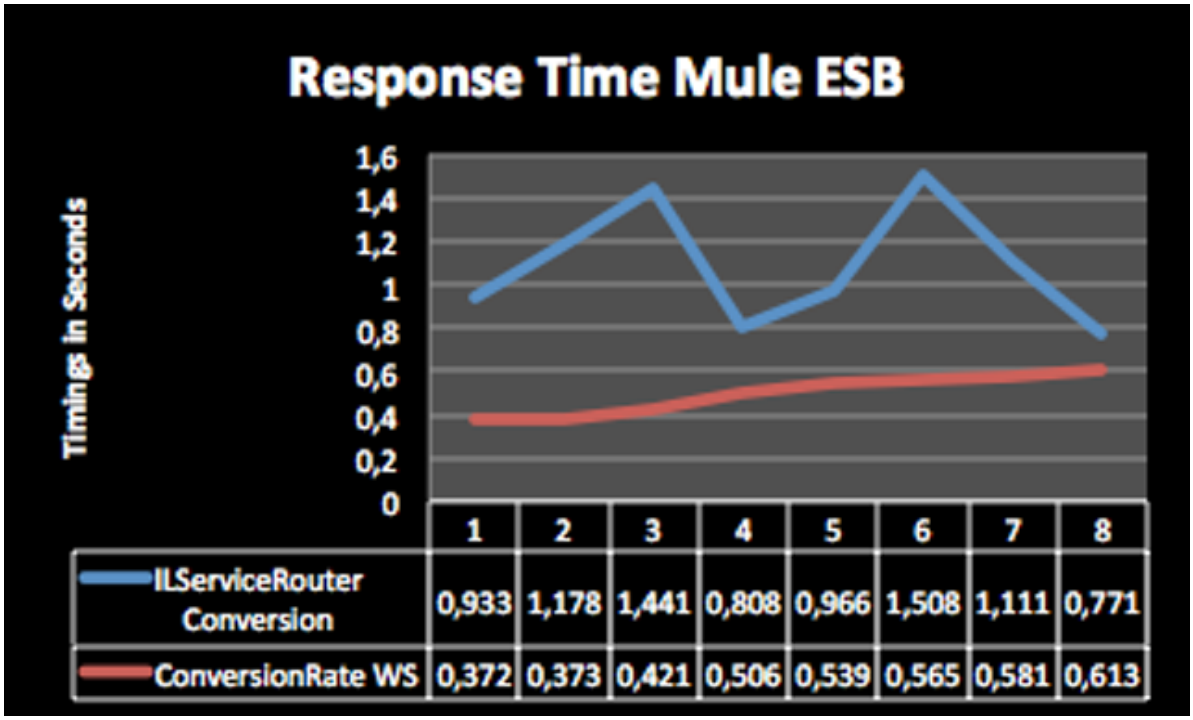

Fig. 6.2: Figure showing results from Mule ESB implementation.

lay is registered in both implementations. One thing to note here though is that the BizTalk implementation was optimized with a much lower polling rate compared to what was standard in the administration console. Therefore from comparing the Tables 6.1 and 6.2, it can be noticed that the BizTalk implementation is much slower. It is important to take this into consideration for usability reasons. If the author had not found the settings to optimize the BizTalk implementation, it would have been drastically slower than the original service and the Mule implementation. Therefore the Mule implementation and software trumps in terms of "out of the box" performance.

As mentioned earlier in this chapter, the Mule tests was run on a computer using wireless network, while the BizTalk tests was run on a computer using wired connection. We can in other words expect better response time when using a wired connection on the Mule implementation. Last, as mentioned, the BizTalk tests were run using a **C#** script. The longer response times observed from the BizTalk implementation is due to the setup being done in the script. If the tests were run similarly to the Mule tests, closer response times would have been observed.

6.3 Platform usability

This thesis is looking into how services can be efficiently integrated into an ESB. From the authors belief there are two important aspects to this. First of all, the ESB software

ConversionRate WS	ILServiceRouter Conversion
00:00:00.372	00:00:00.771
00:00:00.373	00:00:00.808
00:00:00.421	00:00:00.933
00:00:00.506	00:00:00.966
00:00:00.539	00:00:01.111
00:00:00.565	00:00:01.178
00:00:00.581	00:00:01.441
00:00:00.613	00:00:01.508

Table 6.2: Table showing the response times (in Hour:Minutes:Seconds) of the Mule ESB implementation.

needs to be, to some degree, easy to use to efficiently integrate services. Secondly, the ESB system needs to be implemented in some way that the service portfolio efficiently can be extended.

The two implementations presented in this thesis does not excel as ESB systems which can easily be extended. But from the usability aspect, the author has experienced that BizTalk provides a more user friendly experience when creating schemas, mappings, orchestrations and more. Mule provides an "easy-to-use" interface and drag-and-drop functionality, but lacks a well-presented schema editor and functionality at all for creating WSDL-files. BizTalk on the other hand provides functionality with built-in components to publishing ESB implementations as web services through Internet Information Services (IIS).

Efficient integration of services into an ESB might also refer to how fast one can implement an ESB consisting of several services. If this is the case, then Mule might provide a better software. WSDL and schema structure is normally provided by the web service hosters when implementing an ESB. The process of integrating the **ConversionRate** web service into the Mule ESB software was done reasonably fast. If we disregard the lacking schema and WSDL editor, Mule provides a much easier interface which is easy to learn.

There is a definite higher learning curve for creating ESB implementations in BizTalk. This should be mentioned since the author has gone through an introductory course for BizTalk server 2013. The author would not have been able to properly make use of the software without this course, due to the lacking documentation for the software. As mentioned BizTalk offers a greater portfolio of built-in components and editors which are very useful when creating and managing ESB implementations. This is however also the reason why BizTalk can be very inefficient for someone which have no background knowledge of working with integration platforms. Due to the higher learning curve of BizTalk these components might not be as easy to find and use for someone with less background knowledge.

6.4 Extendability

In this section we evaluate the extendability of the two ESB implementations. Both implement a service bus with a small amount of integrated web services which is likely to be extended. To extend these implementations one will have to use either the *Consume Web Service* component in Mule or *Consume Adapter Service* wizard in BizTalk.

In BizTalk this will add generated schemas, multi-part message types and port types. In addition to this, for the implementation in BizTalk one will have to edit the service schema and the orchestration to accompany the functionality provided by the newly integrated web service. Also it will be necessary to create a mapping between the data provided in the service schema to the integrated web service schema, and back to the service response schema. Figure 6.3 shows a screenshot of the service schema structure.

In Mule, the *Consume Web Service* component will add schemas and a port to the web service so it can be invoked. The component in Mule acts as part of the message flow rather than a wizard, but still mappings can be made to the consumed web service schema. To extend the Mule implementation further from the current single integrated service is quite simple, as explained in chapter 5.

When comparing the two implemented prototypes, one can observe that Mule offers quicker extending of the service bus. This is due to the impact of consuming a web service. In BizTalk this adds many files, such as schemas and orchestrations. Due to this fact, many developers choose to create a new BizTalk project to place these files onto. This adds a lot of overhead when wanting to effectively integrate new web services as the service bus is being deployed, and the process of extending the initial service bus becomes very tedious. Mule does not add a bunch of new orchestration files and schema files on the other hand, which makes the integration process much faster.

6.5 Deployment

Deploying the two prototypes implemented in this thesis is performed in two different ways. BizTalk uses the built-in deploy[2] method in *Visual Studio*[3], which is the standard editor for BizTalk applications. This exports the application and its artifacts to the specified SQL server. Otherwise the application can be exported to a Windows installer (.msi) file which is imported directly in the *BizTalk Administration Console*.

The BizTalk prototype will need, together with the Windows installer (.msi) file, a well specified binding file. The binding file will provide different options for databases, as well as provide the binding configuration to the integrated web services. With this configuration file the service can be configured to run on any production system, but it will require some changes.

[2] `https://msdn.microsoft.com/en-us/library/aa561812.aspx` collecte 29.04.15

[3] `https://www.visualstudio.com` fetched 29.04.15

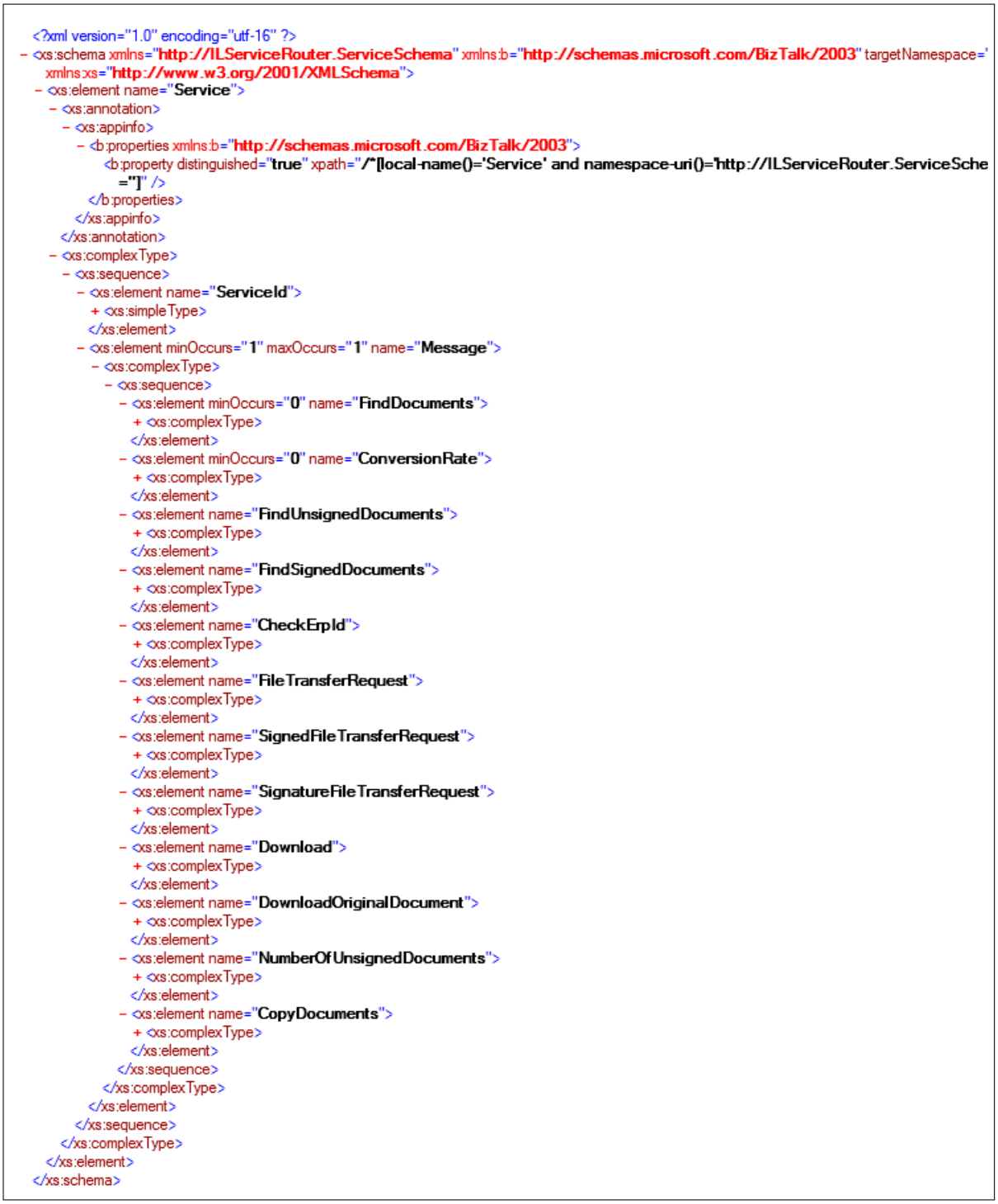

Fig. 6.3: Screenshot of the Service schema structure.

The Mule prototype does not require any reconfiguration to work. The application is exported as a compressed folder and can be deployed on any production system running *Mule Management Console* (MMC). The MMC allows the Mule application to be monitored and analyzed while running. MMC also allows the developer to set up different deployments in different environments.

6.6 Uncompleted work

The two implemented prototypes are currently only being used as experimental implementations. They have not been tested against applications which make use of the web services that have been integrated into the service bus. Therefore validating if the two prototypes satisfies the eSecretary application can not be determined.

The two prototypes does not enforce any encryption, other than what is provided by the integrated web services. The two platforms do offer security in measures of authenticating system users. However the author feels that the implementations might need more encryption to be used in a production environment.

As mentioned in chapter 5, the Mule ESB prototype only integrates the **ConversionRate** web service. Due to time constraints the author was not able to integrate the eSecretary web service. The problem which was encountered, was that the eSecretary WSDL-file did not provide an equivalent defined input to the web service functions. This created a problem where the input defined in the WCF service did not match the input requirements defined in the WSDL-file.

6.7 Improvements for BizTalk application

The prototypes presented in this thesis shows how an ESB can be implemented in Mule ESB and BizTalk Server 2013. The Mule implementation shows promise, but for Jupiter System Partner and future work, we will focus on the BizTalk application since it falls closer to the implementation platform. This is also the preferred platform due to existing knowledge in the enterprise.

This section will be used to discuss possible improvements to the current design and implementation of Information Logistics Service Router.

6.7.1 Performance

As discussed earlier in this chapter, the author was able to optimize the BizTalk implementation to get better performance when routing the request to the integrated web

service. The performance after these optimizations is now comparable to the Mule implementation.

The author has considered of creating a separate cache to enhance the performance of the implementation. If this cache was to be implemented, it would store highly requested data so it could be served faster to the customer. Data could be stored in XML documents, as was done in one of the early experimental implementations (presented in chapter 5), or the highly requested data could be stored in a local database.

This design would however require more storage space on the server. A separate process would also be needed to decide how often data would be updated in the cache. For some applications it is also necessary to get the latest version of data, meaning this design will create a tighter coupling to the web services.

6.7.2 Security

The implementation does not implement any extra security or authentication, though it is a requirement for this type of system. The author knows that there are built-in mechanisms in the *BizTalk Administration Console* to enable authentication.

The author also wants to enable encryption for messages going between the client and service bus, and the service bus and integrated web services. This would ensure best possible security of the ESB implementation and provide security from *man-in-the-middle* attacks[4].

6.7.3 Integrated eSecretary module

As explained earlier in chapter 4, the ESB implementation does not support optimal integration of the eSecretary web service. Due to constraints in the web service functionality, different customers cannot be separated in the web service. So if several companies' documents are being stored in the same database, the service bus won't be able to distinguish the documents from one company to the other.

For this reason the author wishes to create an integrated version of the eSecretary web service in BizTalk with functionality to distinguish between companies. With this integrated version of the eSecretary web service, the performance would also improve.

One part of the eSecretary web service merges signature documents together with invoices. This part is a separate web service created by Jupiter System Partner. The author thinks this functionality could be integrated into the same ESB implementation. This will minimize the number of web services running for one application, and make the functionality available for future applications developed by Jupiter System Partner.

[4] `http://searchsecurity.techtarget.com/definition/man-in-the-middle-attack` (collected 30.04.15.)

However, this would create redundant functionality in the ESB implementation. This design would then not enforce a SOA, since it would create a service bus which is more brittle and exposed to problems.

6.7.4 Service Lookup

Service end-point lookup is used in the paper by Karim M. Mahmoud [3] and in the ESB toolkit developed for BizTalk. This enables the client to look up the web service end-point address, instead of routing the request through the ESB implementation. This could be useful in cases where the client demands lower response times than what the ESB provides. The Service lookup mechanism could easily be added to the implementation by having a table of the available web service addresses and service names.

6.7.5 On-premises vs. Cloud

As mentioned in chapter 3, Microsoft has developed a platform for deploying an ESB in the cloud. This might be very relevant for Information Logistics Service Router since the Azure platform enables dynamic scaling and much easier administration of the deployed services.

Moving the BizTalk implementation to the cloud would be beneficial for Jupiter System Partner as this would discard the need to administrate and monitor a local server. Also purchasing components to cope with needed resources would not be needed, since that is all taken care of by the cloud platform.

6.8 Problem Definition Solved

In chapter 1 we presented the problem definition for this thesis:

> "*The project will focus on integration of several web services, where the end system is an integration platform where web services can be invoked from a singular access point. This master thesis will look into products, such as Mule ESB, BizTalk Server 2013, and IBM WebSphere MQ, which integrate several services to a singular access point. The products will be analyzed against each other and evaluated to determine the best approach for the system.*
>
> *The project will focus on enterprises which constantly expands its web service portfolio. For these enterprises the benefit of having an integration platform which can be efficiently expanded is very important, since it will allow reuse of existing functionality. It is also useful since the web services need only be*

managed inside the integration platform in cases where the individual web services are being moved or upgraded to new versions.

The project will integrate web services such as eSecretary, especially developed for Kraemer Maritime AS[5]*, which is part of the Information Logistics product idea. The resulting product will integrate web services running at customers, into a singular access point using BizTalk Server 2013. However, integrating web services into a singular access point can be time consuming and tedious. This thesis will therefore look at how this process can be streamlined.*

How can services be efficiently integrated in an Enterprise Service Bus (ESB)?"

The remaining part of this chapter will discuss how this thesis has solved the problem definition.

"*The project will focus on integration of several web services, where the end system is an integration platform where web services can be invoked from a singular access point. This master thesis will look into products, such as Mule ESB, BizTalk Server 2013, and IBM WebSphere MQ, which integrate several services to a singular access point. The products will be analyzed against each other and evaluated to determine the best approach for the system.*"

In this thesis we have presented two prototypes, with focus on a BizTalk implementation, which integrates services into a singular access point. In chapter 3 we presented Mule ESB, BizTalk Server 2013 and IBM WebSphere MQ, and discussed how these three compare to each other. In this thesis, the author have implemented two prototypes, one in BizTalk Server 2013 and one in Mule ESB to compare the two in performance and usability. The evaluation of the two products shows that both provide great software for building an ESB. Choosing between these two depends on the developers experience and preferred deployment platform.

"*The project will integrate web services such as eSecretary, especially developed for Kraemer Maritime AS*[6]*, which is part of the Information Logistics product idea. The resulting product will integrate web services running at customers, into a singular access point using BizTalk Server 2013. However, integrating web services into a singular access point can be time consuming and tedious.*"

[5] `http://www.kraemer.no/no/omoss/` (collected 26.05.15.)
[6] `http://www.kraemer.no/no/omoss/` (collected 26.05.15.)

In the presented prototype developed on the BizTalk platform we have integrated the eSecretary web service and the **ConversionRate** web service through built-in functionality in BizTalk Server 2013. The services are routed through an orchestration implemented by the author and the services are made available through a singular access point pointing to the ESB.

> "***How can services be efficiently integrated in an Enterprise Service Bus (ESB)?***"

In this thesis we have presented Information Logistics Service Router - an ESB developed in BizTalk Server 2013 which integrates the eSecretary web service and the **ConversionRate** web service into a singular access point. For comparison we have implemented a prototype in Mule ESB to compare how they differ in performance and usability. This chapter has presented the evaluation of the two software systems. The author has experienced that BizTalk offers more functionality which can be very efficient for developers with previous knowledge. Mule ESB on the other hand, provides a more "easy-to-use" ESB software, which can be easier to use for developers with less background knowledge.

Further this thesis has examined Information Logistics Service Router, and concluded that the implementation does not excel in efficiently integrating services. The reasoning for this is that very many changes need to be made to the implementation to integrate a new service, since the implementation is tightly coupled to the integrated web services. The implementation provides a basis, where the extending of integrated web services can be made quite easily. However, the process of integrating a service can be time consuming.

The author had initially envisioned the system to integrate the web services by using dynamic ports, which could be changed when web service access points where moved. Currently the service bus uses static ports, which cannot be changed dynamically. Therefore if any of the web services are moved, the service bus need to integrate the new web service access point.

The author had also initially thought of integrating one version of the eSecretary web service. However, the ESB implementation will potentially need to integrate several versions due to lacking functionality in separating between different company databases. The author has explained the missing functionality earlier in this chapter.

The author had also envisioned that the ESB implementation would provide response times comparable to directly connecting to the web services. However, the implementation does not achieve this due to overhead from orchestration parsing in BizTalk.

Based on the evaluation and testing of Information Logistics Service Router and ESB building platforms, the author believes that this thesis addresses and solves the stated problem. Both platforms provide an efficient way to integrate web services, but the system implemented in this thesis does not currently allow efficient integration of web

services due to Jupiter System Partner's requirements. However, the underlying architecture allows for the system to be changed to efficiently integrate future web services.

Chapter 7

Conclusion

This thesis has presented a prototype to Information Logistics Service Router - an Enterprise Service Bus (ESB) implemented in BizTalk Server 2013, which integrates the eSecretary web service implemented by Jupiter System Partner and **ConversionRate** web service. The latter is a web service to get the conversion between two currencies.

Information Logistics Service Router has provided Jupiter System Partner with a basis for their product idea Information Logistics. The system provides an integration platform for the eSecretary web service developed by Jupiter System Partner. The system will be used to integrate the existing eSecretary web services distributed at different companies into a singular access point. The research and implementation accomplished in this thesis will be used for future work for Information Logistics Service Router and implementing the remaining components of Information Logistics.

The thesis presented the definition of Enterprise Service Bus (ESB), Service-Oriented Architecture (SOA), and the different platforms for implementing an ESB, as well as differences in functionality and implementation. The thesis has explained the product idea and how the author has gone from not knowing much about ESB and getting an introductory course in BizTalk, to implementing a prototype of the desired system.

The thesis later explained the design ideas for the ESB implementation and presented the final design for Information Logistics Service Router. It presented the specific design ideas for the system, and later the implementation in BizTalk.

The thesis also presented an experimental version of an ESB implemented in Mule ESB to evaluate the differences in performance and usability in integrating web services. Chapter 6 evaluated that the problem definition was accomplished in terms of comparison between ESB building softwares. However, the author feels Information Logistics Service Router does not excel as an integration platform where services can be efficiently integrated.

In the end, the author was satisfied with the results of the project given the limited timeframe of this thesis.

7.1 Concluding remarks

The author was not able to implement and test all the functionality of the ESB, due to the limited timeframe of this project. The author was sent on a introduction course for learning the basics of BizTalk Server 2013. This postponed the implementation startup in this project. The author has mentioned some of the future work of this product in the next section.

Another remark to make note of here, was that the author struggled to find articles and work which could be related to this thesis. From the authors belief, there are many ESB implementations, but few of them integrates web services with different functionality and data. The related work presented in chapter 2 though, especially the paper by Karim M. Mahmoud [3], did resemble the envisioned system.

Finally, for the sake of good order, the author wants to mention that he is currently employed by Jupiter System Partner, which might inherently skew any evaluation involving this company. However, the author is fully aware of this problem and has made every effort to remain objective throughout this work.

7.2 Future Work

As explained earlier in this thesis, the Information Logistics Service Router prototype developed in BizTalk Server 2013 still have room for much future work and improvements.

One important feature which the author wants to be implemented into the system, is a service end-point module as mentioned in chapter 6. With such a module, applications which needs lower response time for the connected web service can collect the end-point address from the ESB implementation and contact the integrated web service directly. The author hopes this feature will be implemented into the system since it might be very relevant for some customers.

Another possible improvement for the implemented system, but which will require more extensive changes to the implementation, is to move the system onto a cloud service such as Azure Microservices. As already mentioned, creating cloud services makes it easier for the developers and owners of the system to scale the system to the number of users. Also it will reduce the administration costs of monitoring the service, making sure that every customer is getting the best experience from the system. The author recognizes that moving the already implemented prototype into a cloud implementation might be time consuming, but hopes it will be considered in the future.

The implemented system does not implement any security measures either. But as explained in the previous chapter, BizTalk Server 2013 implements authentication which can easily be added to the implemented system. This will make sure that only customers

with access to the system can make use of it. This should be a quite simple feature to implement into the system, so the author expects this will be added in the future.

Finally, as mentioned in chapter 5, the author wanted to implement an integrated application version of the eSecretary web service so it could be modified to support several companies in its functionality. If several customers are using the eSecretary web service on a different database, it means that the system will need to be extended with eSecretary web services connected to other databases. The author reckons this part will be implemented in the near future, since it is a crucial part to the future development and sale of the Information Logistics product.

Last as mentioned in chapter 1 this is part of a product idea developed by Jupiter System Partner. There are still many modules which are part of the product idea which can be implemented into BizTalk and other platforms. The author hopes that this thesis will serve as a good basis for the future work of the product idea.

References

[1] Marius Lundblad. How can data be integrated from various systems? *University of Tromsø*, December 2014.

[2] John Callaway Quicklearn. BizTalk Introduction Course. `http://www.quicklearn.com/class.aspx?class=BTSI13`, February 2015. [Online; accessed 29-may-2015].

[3] Karim M. Mahmoud. A Unified Messaging-Based Architectural Pattern for Building Scalable Enterprise Service Bus. *IIWAS '13 Proceedings of International Conference on Information Integration and Web-based Applications & Services, Pages 697*, December 2013.

[4] Ning He and Zoran Milosevic. B2B Contract Implementation using Windows DNS. *Proceedings Workshop on Information Technology for Virtual Enterprises. ITVE 2001, Pages 71-79*, January 2001.

[5] Charles Herring and Zoran Milosevic. Implementing B2B Contracts Using BizTalk. *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, January 2001.

[6] Paul Brebner. Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application. *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on, Pages 404-411*, August 2009.

[7] Ding Kun, Zhang Xiaoyi Deng Bo, and Li Ge. Optimization of Service Selection Algorithm for Complex Event Processing in Enterprise Service Bus Platform. *Computer Science & Education, 2009. ICCSE '09. 4th International Conference on, Pages 582-586*, July 2009.

[8] Mohammad H. Danesh, S.M. Amin Kamali Bijan Raahemi, and Greg Richards. A Distributed Service Oriented Infrastructure for Business Process Management in Virtual Organizations. *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on, Pages 1-6*, April 2012.

[9] David Frank, Liana Fong Linh Lam, and Christopher Vignola Ru Fang. An Approach to Hosting Versioned Web Services. *Services Computing, 2007. SCC 2007. IEEE International Conference on, Pages 76-82*, July 2007.

[10] Thorsten Heller (via Greenbird Integration Technology AS). SOA in practice. The sibling rivalry between ESB and BPE. `http://www.slideshare.net/toheller/soa-in-practice-the-sibling-rivalry-between-esb-and-bpe`, July 2010. [Online; accessed 22-may-2015].

[11] Information from Arne Seime at Greenbird Integration Technology AS.

[12] Sam Vanhoutte. Azure BizTalk Microservices, initial thoughts. `http://www.codit.eu/blog/2014/12/03/azure-biztalk-microservices-initial-thoughts/`, December 2014. [Online; accessed 22-april-2015].

[13] Peter Borremans. Azure Microservices – first glance. `http://www.codit.eu/blog/2014/12/05/azure-microservices---first-glance/`, December 2014. [Online; accessed 22-april-2015].

[14] Microsoft Corporation. Microsoft Integration Platform. `http://azure.microsoft.com/nb-no/documentation/infographics/integration/`, 2015. [Online; accessed 22-april-2015].

[15] Microsoft Corporation. Microsoft BizTalk ESB Toolkit. `https://msdn.microsoft.com/en-us/library/ff699598.aspx`, 2014. [Online; accessed 22-april-2015].

[16] Eric Knorr. 2004: The Year of Web Services. `http://www.cio.com/article/2439869/web-services/2004--the-year-of-web-services.html`, 2003. [Online; accessed 07-may-2015].

[17] Anne Thomas Manes. Book Excerpt: When to Use Web Services. `http://www.computerworld.com/article/2566429/app-development/book-excerpt--when-to-use-web-services.html`, 2004. [Online; accessed 07-may-2015].

[18] JBoss Inc. Why ESB and SOA? `http://docs.jboss.org/jbossesb/whitepapers/WhyESB.pdf`, 2006. [Online; accessed 03-may-2015].

[19] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating Web Services on the World Wide Web. *WWW '08 Proceedings of the 17th international conference on World Wide Web, Pages 795-804*, April 2008.

[20] Abdelghani Benharref and Salah Bouktif Mohamed Adel Serhani. A Managerial Community of Web Services for Management of Communities of Web Services. *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on, Pages 97-104*, June 2010.

[21] Shahab Mokarizadeh and Mihhail Matskin Peep Küngas. Utilizing Web services Networks for Web service Innovation. *Web Services (ICWS), 2014 IEEE International Conference on, Pages 646-653*, June/July 2014.

[22] Khalid Elgazzar and Patrick Marting Ahmed E. Hassan. Clustering WSDL Documents to Bootstrap the Discovery of Web Services. *Web Services (ICWS), 2010 IEEE International Conference on, Pages 147-154*, July 2010.

[23] M.K. Zand and M.H. Samadzadeh. Software reuse Issues and perspectives. *Potentials, IEEE (Volume:13 , Issue: 3), Pages 15-19*, August/September 1994.

[24] MuleSoft Community. What is an ESB? `https://www.mulesoft.org/what-esb`. [Online; accessed 21-may-2015].

[25] Harry M. Sneed and Stephan H. Sneed Chris Verhoef. Reusing Existing Object-oriented Code as Web Services in a SOA. *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the, Pages 31-39*, September 2013.

[26] Min-Sheng Hsieh and Ewan Tempero. Supporting Software Reuse by the Individual Programmer. *ACSC '06 Proceedings of the 29th Australasian Computer Science Conference - Volume 48, Pages 25-33*, January 2006.

[27] Charles W. Krueger. Software Reuse. *ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 24 Issue 2, Pages 131-183*, June 1992.

[28] Lombard Hill Group. Software Reuse 101: What Is Software Reuse? `http://lombardhill.com/articles/software-reuse-101-what-is-software-reuse/`. [Online; accessed 26-may-2015].